ASPECTS OF N-TUPLE CHARACTER RECOGNITION FOR A BLIND READING AID

by

John Anthony Nappey

A thesis presented for the degree of
Doctor of Philosophy in the Department
of Electrical Engineering and Electronics,
Brunel University.

September 1977

ABSTRACT

This thesis reports research conducted into a character recognition system suitable for use in a reading aid for the blind. A brief review of blind reading aids is given, showing the need for a device which is cheap, simple and effective. The structure of a proposed reading aid fulfilling these needs is outlined, with a list of the desired characteristics of each of its subsystems.

The remainder of the thesis is concerned with research into just two of these subsystems: the input device and the character recognizer. A detailed review of pattern recognition by the n-tuple method is presented, followed by a description of the experimental techniques used in obtaining real data from a camera system, and in simulating various recognizer structures. The camera system and computer programs developed specifically for the research are described in detail.

Several series of experiments are reported, concerned mainly with investigating problems associated directly with the blind reading aid, namely accommodation of multifont printed text and of the tracking errors inherent in data from a hand-held probe. A further series of experiments, aimed at improving the performance of the recognizer within fixed size constraints, i.e.,optimisation, has a wider field of application.

Finally suggestions are made as to how the recognizer might be implemented in a reading aid, using RAMs, ROMs, or PLAs as the main storage elements.

CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Blind Reading Aids

1.1 Introduction

This thesis reports research conducted into a character recognition system suitable for use in a reading aid for the blind. The bulk of Chapter 1 deals with the history of blind reading aids, showing the need for a device which is both simple (and thus cheap), and easy to use. The structure of a proposed reading aid fulfilling these needs is then outlined, with a list of the desired characteristics of each of its subsystems. Later chapters deal exclusively with just two of these subsystems: the optical input device, and the character recognizer. Details are given of the experimental techniques employed in obtaining real data from a camera and in simulating various structures of character recognizer. The design of both the camera and the simulation programs are discussed, followed by a record of the experiments performed both in investigating the capabilities of the recognizers, and in attempting to improve their performance. Finally a summary of the research is presented, showing the areas in which definite advances are felt to have been made, together with suggestions for future lines of research.

1.2 Review of Blind Reading Aids

As suggested by its name, a blind reading aid is a device which enables a blind person to 'read' a book, or other printed source of information, without the aid or intervention of a sighted person. The most widespread forms of aid are the 'talking book', which is simply a tape recording of a sighted person reading the book aloud, and Braille, which uses a complex alphabet whose individual characters are embossed on a suitable material so that the blind person may read them by touch. Both of these methods suffer from the fact that all reading material must undergo expensive processing before being available to the blind user, with the result that the range of material is rather limited. Braille has the further

disadvantage that a long training period is required, especially when the user has become blind late in life.

There thus exists a very real need for devices enabling blind people to read normal printed material, without the need for special processing and preferably entailing a minimal amount of training. Attempts to produce such devices have been made over the last sixty-five years, falling into three broad categories, according to their degree of sophistication: direct translation aids, speech output aids, and quasi-speech output aids.

1.2.1 Direct Translation Aids

Direct translation aids simply translate the optical image of the text to be read into another medium, sound or touch, which is more accessible to the blind user. The best known of these devices is the Optacon (1) which has a hand-held photomatrix scanner driving a corresponding matrix of piezoelectric transducers capable of stimulating, by vibration, an area of the user's finger. Thus the image of the printed text becomes a pattern of vibrating and non-vibrating areas. This very simple arrangement is versatile in that it is immediately applicable to a wide variety of input texts, but suffers from the inherent fault of all direct translation aids, i.e., the lack of any processing in the machine leaves the whole burden of recognition to the operator. This is very important when the text is being translated into a form which is unnatural to the user, as it entails usually a very long training period, at the end of which the blind person is still able to read only very slowly. It is understandable that such conditions can be discouraging to a new user. For example, in the case of the Optacon, reading speeds of upto 30 words per minute (wpm) are reported after 50 hours training (2). In comparison, someone skilled in Braille is able to read at up to 180 wpm (3). Other direct translation aids have achieved better results than the Optacon, by using the blind person's sense of hearing. The Visotoner and Lexiphone (4) both produce a series of coded chords whose structure

depends on the distribution of black ink currently being detected within the scanned line of text by a linear photosensor array. This means that typically three chords must be assimilated before a single letter can be recognized, but nevertheless speeds of up to 40 wpm have been achieved. The first successful reading machine produced, the Optophone, in 1913 (5), worked on a similar principle, but achieved only 10 wpm after 100 hours of training.

1.2.2 Speech Output Reading Aids

Lying at the other extreme of sophistication are the reading machines which scan the printed text, either automatically, or with a hand-held probe, perform character recognition on this input, and then output a spoken version of the text. Many approaches to this task have been attempted, usually the only significant differences between which being in the method of speech output used. The simplest approach is the triggering of a prerecorded spoken letter memory (e.g., on a magnetic drum) whenever the character recognizer makes a decision on the letter currently being scanned. The result is a very disjoint form of spelled speech, with little or no flow between successive letters. One step better is the production of the spelled speech from digitally stored parameters (e.g. phonemes), thus allowing consecutive letters to flow into one another by altering the durations of the sounds (e.g., by extending or repeating them until the character recognizer makes its next decision). Even better are those machines which generate synthesized speech, outputting a syllable, or even a word at a time, usually involving a look-up process in a dictionary of common words, with the default of spelled speech for those words not recognized (6). Needless to say, such machines require massive amounts of storage and computational power, and have so far only been projected, or at best only simulated in part.

Examples of 'recognition plus spelled speech' machines (or simulations rather), are the Cognodictor (7), which aims at a speed of 80 wpm, using

an optical memory for generating the spelled speech, and the project supported by the MIT (8), which has achieved speeds of 60 wpm after only 1 hour's training on the simulation (6). It is suggested that the upper limit on the speed at which spelled speech may be assimilated is around 200 wpm (9). A discussion of the synthesis of continuous speech as related to blind reading aids may be found in (10).

1.2.3 Quasi-Speech Output Aids

Lying between these two extremes are those reading aids which do not perform a recognition task on the printed input, but provide a translation into an audio code. The code is chosen specifically to resemble natural speech in that it has similar characteristics as regards formant content, etc. In the machine proposed and simulated by Uttley (11), particular geometric features, e.g. corners, crossings, etc. of each input letter are recognized (without recognizing the letters themselves), and are translated into audio features which are known to be characteristic of speech. The results of initial tests have proved encouraging.

1.2.4 Summary

In summary, direct translation reading aids are attractive in their simplicity and cheapness, but perform badly in the more important respects, i.e., reading speed and ease of learning. Machines which provide a translation into an invented language resembling speech would appear to offer greater advantages, and should be comparable in efficiency to a spelled speech machine, with the proviso that the blind user must learn a totally new language first. The success of such a machine will thus be critically dependent upon the choice of output language. It is reasonable, then, to assume that a machine producing some form of immediately recognizable speech would be the most readily acceptable. Since such a machine must contain a character recognizer as well as the speech output device, it is important to maintain a low level of

complexity in order to keep the cost reasonable, so it would appear that a spelled speech output derived from a digital store would form the best compromise. The following section outlines the structure and use of a suggested reading aid employing a character recognizer linked to a spelled speech utterance generator.

1.3 Outline of the Blind Reading Aid

The blind reading aid envisaged consists of four distinct subsystems: a camera, a character recognizer, an utterance generator, and the blind user. The relationships between these are shown in Fig.1.1.

(a) The camera at any given time is sending a digital picture of a small area of the source text. A convenient way to enable manual scanning is to connect a flexible light pipe to the camera, thus giving an easily manipulated lightweight probe. Consistent illumination can be ensured by connecting some of the fibres of the light pipe to a light source.

(b) The character recognizer inspects the picture presented by the camera, and if a recognizable character, i.e., letter of the alphabet, is visible, outputs its decision as to what character is present.

(c) The utterance generator, when triggered by a decision from the recognizer, outputs an audible utterance related to the class of the input character. For example, a phonetic version of the letter 'G' would be output whenever the recognizer makes the decision: 'Class G'.

(d) The blind user completes the loop between the subsystems. He is listening to the output from the utterance generator, and simultaneously manipulating the input probe of the camera. This has the great advantage over many previous reading aids that the user has absolute control over what is being'seen'by the system. This helps in several ways, firstly in that he may read the text at whatever speed he chooses, with the ability to go back and

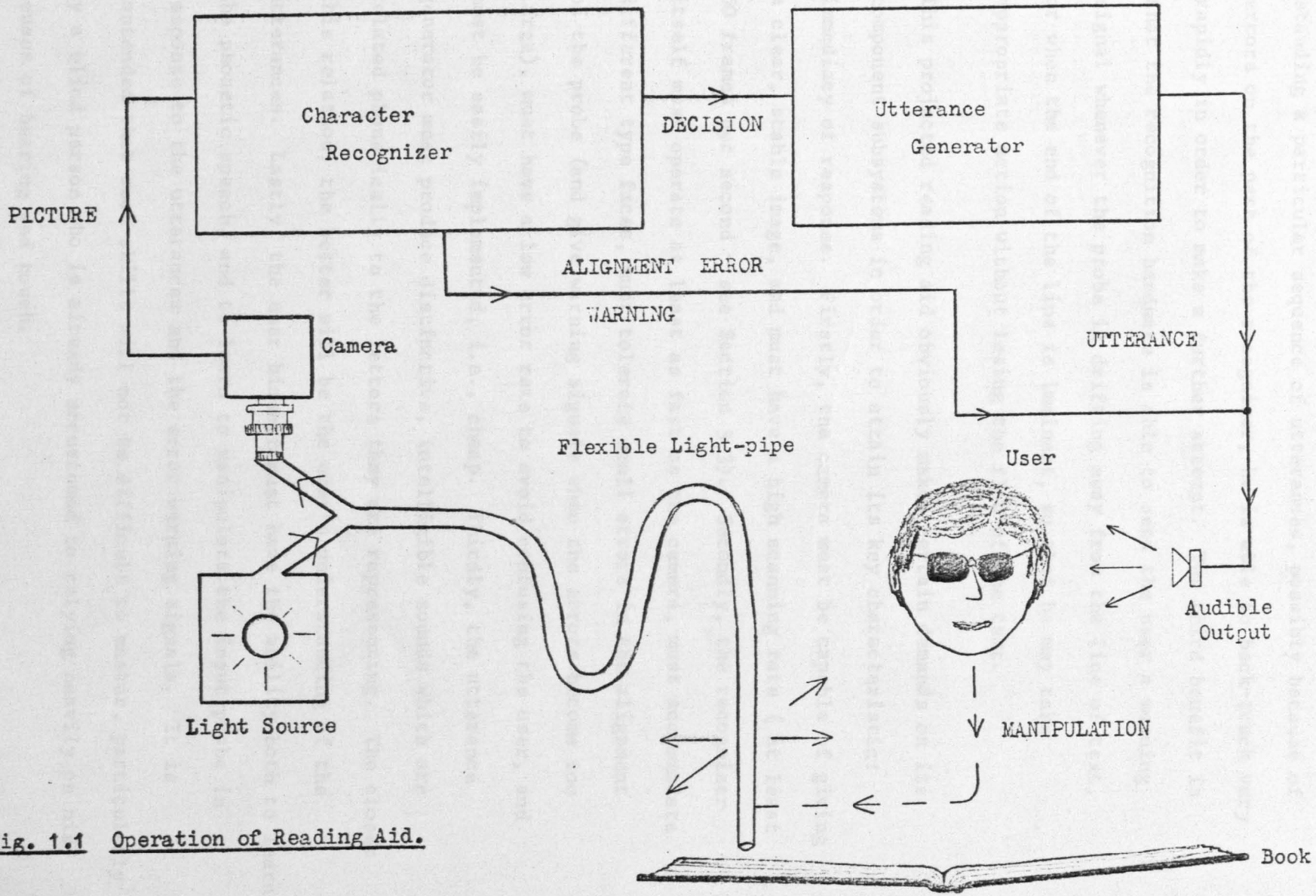


Fig. 1.1 Operation of Reading Aid.

re-read passages. Secondly, should he experience trouble in understanding a particular sequence of utterances, possibly because of errors on the part of the recognizer, he is able to back-track very rapidly in order to make a further attempt. The third benefit is that the recognition hardware is able to send the user a warning signal whenever the probe is drifting away from the line of text, or when the end of the line is imminent, so that he may take appropriate action without losing the flow of the text.

This projected reading aid obviously makes certain demands on its component subsystems in order to attain its key characteristic: immediacy of response. Firstly, the camera must be capable of giving a clear, stable image, and must have a high scanning rate (at least 50 frames per second - see Section 5.3). Secondly, the recognizer itself must operate at least as fast as the camera, must accommodate different type faces, must tolerate small errors in the alignment of the probe (and give warning signals when the errors become too large), must have a low error rate to avoid confusing the user, and must be easily implemented, i.e., cheap. Thirdly, the utterance generator must produce distinctive, intelligible sounds which are related phonetically to the letters they are representing. The closer this relation, the better will be the user's understanding of the utterances. Lastly, the user himself must have the ability both to learn the phonetic speech, and to learn to manipulate the input probe in response to the utterances and the error warning signals. It is contended that such skills will not be difficult to master, particularly by a blind person who is already accustomed to relying heavily on his senses of hearing and touch.

Detailed studies of phonetic languages and experimental designs for the utterance generator have been made by Greenshields (12). The remainder

of the present thesis is concerned with the character recognizer, giving special attention to the desirable attributes listed above.

1.4 Summary

A brief history of research into blind reading aids has been presented, concluding that a useful device could be produced at low cost with definite improvements over the devices generally available at present. An outline of the envisaged reading aid was given, with particular emphasis on the areas requiring detailed study, namely the character recognizer and the utterance generator. Subsequent chapters are concerned only with research into the former of these.

Chapter 2

The n-tuple Method

2.1 Introduction

Chapter 1 described the modus operandi of the envisaged reading aid for the blind. An important part of this device is the character recognizer, and the present chapter is concerned with the choice of pattern recognition scheme for the recognizer. Section 2.2 constitutes a detailed review of previous work on the n-tuple method, emphasising those aspects which are particularly relevant to the present research, whilst Section 2.3 shows how the method satisfies the requirements listed in Chapter 1. Sections 2.4 and 2.5 outline the characteristics of the simulated recognizer and the experiments performed thereon.

The n-tuple method is a scheme of pattern recognition in which many features, called n-tuples, are selected arbitrarily from elements of the vector representing the input pattern. During the training phase, a representative set of patterns of each class is presented to the recognizer, which stores the frequency of occurrence of each state of each n-tuple, according to class. In all the experiments reported, the pattern vector has been binary, i.e., each element is represented by a single bit. Thus the state of a particular n-tuple may be expressed as a binary word n bits long, as in Fig. 4.3 (section 4.4.2), where n-tuple 0 is in the state 0101, n-tuple 1 is 0010, and n-tuple 2 is 0110. A further simplification applied in most cases is that only approximations to the real occurrence frequencies are stored, usually to the extent of allocating only one bit per n-tuple state.

After training, the recognizer has built up for each class an approximate occurrence probability for the state of each n-tuple. Suppose it is now desired to test the performance of the recognizer,

and a previously unseen pattern is presented to it. As before, each n-tuple assumes a particular state, for which the probability of occurrence is now known (albeit approximately). The response phase of the recognizer involves combining, for each class separately the stored probability of the current state of each n-tuple. Thus for each possible class one obtains the joint probability of occurrence of the current set of n-tuple states, i.e., of the current input pattern. The application of Bayes' rule (c.f. Chapter 3 of (13)), leads to the choice of the class whose joint probability value is the largest as the most likely class to which the current pattern belongs.

It can be appreciated from the above description that the n-tuple method offers much scope for experimentation, since there are many inter-related parameters, namely, n-tuple size; number of bits allocated to each n-tuple state; method of selection of the n-tuples (known as the 'mapping' of the pattern elements onto the n-tuples); size and nature of the training set; and the way in which the stored probabilities are combined to give the joint probability for the whole pattern. A further refinement is the derivation of a confidence value for a particular decision, based on the magnitude of the final joint probability: patterns causing low-confidence decisions are rejected.

2.2 Review of the n-tuple method

The n-tuple method was first introduced by Bledsoe and Browning (14), whose initial experiments involved the use of handwritten alphanumeric characters. The quality of this data base was questioned by Highleyman and Kamentsky (15), but nevertheless the results obtained were significant. Bledsoe and Browning used a pattern matrix of 10 x 15 bits randomly mapped onto 4-tuples to show that discrimination could be achieved between 36 classes whilst using only a single bit per class

to store the occurrence frequency of each n-tuple state. Under such a scheme, a '1' is stored in the appropriate single bit location if the corresponding n-tuple state is ever experienced during the training phase; otherwise a '0' is retained. During the testing phase, a response is obtained for each class simply by summing the contents of the locations corresponding to each of the current n-tuple states. The class to which the current pattern belongs is deemed to be that which gives the largest response. Experiments were performed to show that recognition performance was related to both training set size, and n-tuple size. For any particular n-tuple size, the performance at first increased with the number of training patterns, reached a peak, then began to fall again. Furthermore, larger n-tuple sizes gave higher performance peaks, but only after a larger number of training patterns. Further experiments indicated that the use of more than one bit in which to store the occurrence frequencies leads to increased performance, but that the choice of random mapping between the pattern and the n-tuples has no significant effect.

The idea of using more than one bit for the storage of state occurrence frequencies was developed more fully by Bledsoe and Bisson (16), who ran a series of experiments in which the exact frequencies were stored. They also investigated the use of various normalizations of the frequency memory matrix, comparing the performance figures over fixed sets of test patterns. The need for such normalizations when frequencies are being stored was demonstrated by Highleyman (17). The most successful normalization was found to be the 'maximum likelihood method', based on work by Minsky (18). The significant feature of this method is that the individual n-tuple state probabilities are multiplied together to give the joint probability for the whole pattern (in fact, the logarithms of the probabilities are stored, and added together to give the logarithm of the joint

probability). Bledsoe and Bisson at the same time reported some small-scale attempts to optimise the choice of n-tuples, i.e., the mapping, by giving priority to those pattern elements which have a high probability of being set to '1'. The reasoning behind this choice is not clear, since it would appear that the optimum n-tuple selected on this criterion would be sampling permanently dark areas of the pattern matrix, and would therefore have no discriminating ability whatsoever.

The first theoretical treatment of the n-tuple method was made by Steck (19), who represented the patterns, the mappings, and the memory matrices by binary vectors, and the processes of training and testing by operations on these vectors. His model was concerned only with single-bit state occurrence frequencies. One of his more obvious conclusions was that the degradation in performance with increasing training set size, reported by Bledsoe and Browning, is due to the 'saturation' of the memory by the increasingly likely occurrence of 'rogue' states of n-tuples, which occur only infrequently. This underlines one of the drawbacks of the simple (one bit per state) n-tuple method: no differentiation is made between commonly occurring states and rare ones. Steck (19) suggested that although the storage of 'real' probabilities undoubtedly improves the performance, it does so at the cost of extra storage which could conceivably be better utilised in making the n-tuples larger, since this too improves the performance.

He developed his model further by making a series of assumptions about the nature of the patterns to be used, in particular that the n-tuples are independent, that all the n-tuples have equal discriminating powers, and that all pairs of pattern classes are equally easy to discriminate between. On the basis of these assumptions (with

admitted inadequacies), Steck derived formulae for the prediction of recognition performance in terms of the variability of the patterns themselves. This variability was itself expressed in terms of the contents of the recognizer memory after exposure to the training set.

A somewhat different approach was taken by Roy and Sherman (20) in their treatment. They considered the maximum likelihood version of the n-tuple method, showing that it yields the best available approximation to the true probability distributions. Furthermore, they showed that an n-tuple recognizer implements decision surfaces in the pattern hyperspace which are equivalent to polynomial functions of order 'n', but with some terms removed (it is a degraded Φ machine). Thus an n-tuple machine can be more powerful than a linear machine of similar size, whose decision surfaces would be simple hyperplanes. Its limitation, however, is that the initial choice of n-tuples determines immediately the form of the decision surfaces, and these are not necessarily the optimum ones for the task in hand.

A series of experiments using the n-tuple method for multi-font recognition of typed numerals was performed by Ullmann and Kidd (21). They made comparisons between the maximum likelihood version and the non-weighted version (i.e. one bit per state), finding that the latter, at least for the restricted pattern sets available, gave the best performance.

Ullmann (22) performed a more thorough series of experiments on the n-tuple method, this time using handwritten numerals. His investigations of the variation in recognition performance with n-tuple size and training set size essentially confirmed Bledsoe and Browning's initial findings (14) for the non-weighted method. Comparison of this

with the maximum likelihood method confirmed earlier findings that the latter yields slightly worse performance. He investigated this phenomenon by counting the occurrence frequency of each state of each n-tuple, finding that for large n-tuple sizes many states were occurring only once, and that under these circumstances the non-weighted method gave the better approximation to the real probability distribution.

Further work was reported by Ullmann (23), this time in an attempt to reduce the storage requirements of the n-tuple method by random superimposed coding, which was a set of random logical operations performed on the contents of the memory after training. The result was a 4-fold reduction in space, but with a great reduction in the simplicity of the system, and a corresponding decrease both in its flexibility, since the addition of further training patterns would be difficult, and in its performance.

Aleksander and Albrow (24) suggested a way of implementing the unweighted n-tuple method using readily available microcircuit memory elements: each n-tuple constituted the address input of a random-access memory element (RAM), so that a particular state of an n-tuple would address a unique location (bit) within a RAM, in which a '1' would be written during training, and which would be read during testing. The concepts of generalisation and discrimination were developed for a two-class recognizer (a dichotomizer) in terms of the correct classification and the correct rejection respectively of previously unseen patterns. A correct classification was defined as being a positive response from all n-tuples of one class simultaneously. The effects of introducing response thresholds were also discussed, and simulation experiments performed on sets of handwritten numerals.

Further development of the idea of generalisation was reported by Aleksander (25), together with an analysis of the recognition process in terms of the Hamming distances between the current pattern being tested and each of the members of the training set. Experiments on handwritten numerals confirmed the general conclusions of the analysis.

Improved prediction of the performance of an n-tuple recognizer is possible by using a more detailed Hamming distance analysis presented by Stonham (26), who considered in addition the Hamming distances between individual members of the training set.

Refinements to the n-tuple method enabling it to recognize sequences of events were reported by Reeves (27). He was concerned mainly with two-class discrimination between simple geometric figures such as circles, squares, and triangles, using an adapted form of the n-tuple method to learn to trace the outline of each shape. The sequence of incremental tracking commands was used as the input pattern to the recognizer, which was made sensitive to sequences by feeding back some of the n-tuple responses (i.e., the single-bit probability values) to the input matrix. Reeves experimented with several novel forms of recognizer structure, by re-defining the function of the constituent memory elements. The basic form of these elements was called a SLAM (Stored Logic Adaptive Microcircuit), which was functionally identical to a random access memory. His first experiments used the 'probabilistic SLAM', in which only a randomly chosen proportion of the accessed memory locations were set for a particular training pattern, the idea being to reduce the sensitivity of the dichotomizer memory to the last training pattern seen. The 'CL SLAM' (Cumulative Learning SLAM) was simply a weighted n-tuple memory element in which four bits were allocated to each n-tuple state. The 'TR SLAM' (Ternary SLAM) stored only two bits per n-tuple state, the four possible states of each location being assigned to the conditions: not

trained, trained to 1, trained to 0, trained to 0 and to 1. Again this was designed to overcome some of the limitations of dichotomizers, which have to be trained to 1 on one class, and trained to 0 on the other class. In evaluating the response of a set of TR SLAMs to a new pattern, only those n-tuples responding with 'trained to 1' or 'trained to 0' would be taken into account.

Cheung (28) gave a useful overview of the various approaches to pattern recognition, and in particular compared the n-tuple method with several other schemes. A series of experiments led him to the conclusions that the rate of filling of the n-tuple state memory is related to the average density of the training patterns (reaching a maximum when the average density is 0.5), and the optimum performance of the recognizer is achieved when its memory is half-full. The experiments used sets of handwritten numerals which exhibited a large degree of variation within classes. From the half-full criterion, Cheung argued that the optimum training scheme for a recognizer would involve training each class separately until half-filled, and that this could be achieved most readily by using a fairly small n-tuple size, to avoid the use of a very large training set. He performed further experiments using a modified form of SLAM whose output was weighted by a factor dependent upon the number of bits set within the SLAM. He obtained the best results using an 'antiramp' function, in which SLAMs containing few set bits were weighted most (thus favouring the most frequently occurring states), and a 'peak' function, in which half-full SLAMs were weighted the most.

Cheung made some attempts to optimise the pattern to n-tuple mapping, by restricting the elements connected to a particular n-tuple to a small area. He found that this caused large differences in the rates

of filling of different SLAMs, (because of variations in 'activity' in different areas of the patterns), and negligible increase in overall performance. His final experiments involved the 'FO SLAM' (Frequency of Occurrence SLAM) which was yet another approach to a weighted n-tuple system, by storing the real occurrence frequencies of the n-tuple states.

True multi-class discrimination by the n-tuple method was reported by Stonham (29), who used the unweighted version to classify mass spectral patterns falling into 28 classes. He found that the criterion for optimum performance suggested by Cheung - half-filling the memory - was not applicable to the case of mass spectra. He attributed this to the much smaller degree of variation exhibited by his patterns. Stonham experimented with two forms of optimisation, both involving the adjustment of existing mappings (in the light of their performance on a fixed set of patterns), rather than by the generation of completely new mappings. Firstly he identified pattern elements which were invariant, i.e., permanently at '0' or at '1', and ensured that they were not connected to any n-tuple, since they could not be of any discriminatory value. The second optimisation involved the identification and re-connection of redundant n-tuples, i.e. those which always gave the same response for all classes, and hence were not discriminating. These two approaches yielded excellent results when applied to recognition of patterns within the test set, but their benefits were less obvious for the recognition of unseen test patterns (termed 'prediction' by Stonham). This was attributed to a lack of sufficiently large and representative pattern sets.

A review of the applications of n-tuple techniques, particularly in artificial intelligence, was presented by Aleksander, Stonham and Wilson (30). In particular, details were given of how a trainable

n-tuple recognizer (using RAMs) may be converted to a test-only recognizer, using read-only memories (ROMs) once the system has been adequately trained. Further refinements of n-tuple recognizers were suggested by Aleksander and Hanna (31), in particular details of the discriminator concept. A discriminator is a set of n-tuples (memory elements) trained to one particular class of pattern. Thus there are as many discriminators as there are classes, and the response of each to a new pattern is simply the sum of the constituent n-tuple outputs (whether they be single bits or weighted outputs). The decision of the recognizer is based on the discriminator having the largest response.

A practical application of the n-tuple scheme was described by Fairhurst and Stonham (32), who used computer simulations of a 4-tuple recognizer to classify machine-printed alphanumeric characters. They found that the greatest improvements in recognition performance were obtained on the removal of spurious noise due to the encroachment of adjacent characters during digitizing and to the occurrence of isolated black bits within the patterns. They also reported significant improvements when the training procedure was modified to include only those training patterns which would otherwise be misclassified. Fairhurst and Stonham also discussed at length various hardware implementations of the recognizer, using RAMs and ROMs, with the respective advantages of adaptability and cheapness.

A completely independent set of experiments on similar data to the above was reported by Williams (33), in connection with automated postcode reading. The experiments involved the use of different random mappings, simple systematic mappings based on pattern element occurrence frequencies, transformations on the patterns (e.g., line

thickening and thinning), and buckpassing. In the latter, a fairly high response threshold was applied at the initial decision, and all rejected patterns were transformed, then passed on to a second recognizer, trained on a set of transformed patterns. The best performance figures were obtained for a multi-level buck passing scheme, in spite of the fact that little effort appeared to have been made to select suitable transforms. Furthermore, it is not clear whether any attempt was made to ensure that the training sets were of the correct size in the main series of experiments using 12-tuples. A minor series using 6-tuples was conducted, with a training set size estimated from Ullmann's experiments (22), which were in fact based on data of an entirely different kind - hand written numerals.

Prorok (34) presented a thorough treatise on n-tuple pattern recognition, deriving analytical forms for the discriminant functions (probability density functions) based on both decision theory and information theory (treating the n-tuples as independent communication channels). He also investigated the variation in recognition performance with various random structures, finding that the probability of obtaining a highly performing random mapping is very small, since the frequency distribution of map performance is sharply peaked. Prorok also reported a series of experiments in which the original random mappings, as well as mappings possessing certain topological properties, were subjected to progressive modifications, with the aim of improving the overall performance. He emphasised, however, that his results were directly applicable only to the particular data set of handwritten numerals available to him, and that further research was necessary in order to establish their generality. Finally he presented a discussion of ways of implementing the n-tuple method, using digital memory elements to store approximations to the true discriminant functions, and deriving a multiple threshold scheme

rivalling in performance the maximum likelihood method.

A further important achievement of Prorok was that his mathematical treatment covered the important case of the Perceptron - an analogue device previously studied in great depth.

2.3 Choice of Character Recognizer

In Section 1.3 were listed the attributes required of a character recognizer suitable for the blind reading aid, namely high operating speed, multi-font capability, alignment error tolerance,low error rate, and cheapness. The n-tuple method, described in detail in Section 2.1, immediately satisfies the first and last of these requirements since it is easily implemented as a parallel, and hence fast network of read-only memories or programmed logic arrays. Multi-font capability and alignment error tolerance may be regarded as two aspects of the same feature: tolerance of pattern variation within classes. The ability of the n-tuple method to cope with this variation is investigated in this thesis, as well as ways of optimising its performance in this respect. Low error rate has always been an important criterion in any character recognition scheme, and the n-tuple method has proved to be certainly no worse than any other general-purpose scheme, but with the additional advantage that it is easily adapted to handle new types of data.

2.4 An Outline of the Simulation

The starting point for the simulation of an n-tuple character recognizer was Bledsoe and Browning's initial configuration, in which only one bit is assigned to each n-tuple state within each discriminator (see Section 2.1.) This was chosen since it offers the most economical recognizer in terms of storage, and although its recognition powers are based on simple approximations to the true probability density functions, it is still capable of achieving reasonable performance . A second reason for choosing a single bit per state was that the majority of previous results are directly applicable, since few workers have made practical use of multiple-bit locations: Bledsoe and Bisson (16), Ullman (21),(22), and Reeves (27).

The n-tuple size adopted for most of the experiments was 8, since this seemed to offer a good compromise between storage space and recognition capability (see Section 2.2). More flexibility was employed in the use of mappings, since this area was to receive the greatest attention, as reflected in the facilities provided by the simulation programs, which are discussed in more detail in Section 4.4.3.

Since the development of a hand-held camera probe was not feasible in the time available*, it was necessary to use a tripod-mounted solid-state camera pointing down at a table upon which the reading material was moved in a simulation of manual scanning. Details of this system may be found in Section 4.2. The pictures derived in this way were generally stored as computer files to form a fixed data-base for the character recognition experiments, though frequent recourse was made to the camera, in order to double-check the verisimilitude of the data.

2.5 An Outline of the Experimentation

2.5.1 Variation of patterns within classes

The experiments performed under this heading are described in Sections 5.2 and 5.3, and include the adaptation of the n-tuple method to cope with variations in type face, in case (upper case and lower case), in position and orientation, and also to segment running printed text into its constituent letters. The latter was achieved by relying on response thresholds to reject unclassifiable patterns, such as those arising when two letters are each only partially in view. These experiments were performed to satisfy the immediate requirements of the blind reading aid, and although they are common to most programmes of

* although one has since been made commercially available.

character recognition research, such experiments have not all been adequately performed in the past for the n-tuple method. Hence it is felt that the knowledge presented in this thesis was not available prior to the execution of the reported research.

2.5.2 Optimisation

Section 5.4 discusses the several forms of optimisation that were investigated, mainly by adjustment of the mapping between the pattern matrix and the n-tuples. Amongst the approaches tested were the building-up of n-tuples from selected pattern elements, the compilation of mappings from selected n-tuples, and the combination of different mappings to resolve confusion between particular pattern classes.

The optimisation experiments were motivated by a desire to improve on the purely random mappings which had hitherto always yielded the best results. Stonham (29) and Stonham and Aleksander (35) described optimisation tests on a recognizer for mass spectra, by the identification and reconnection of invariant pattern elements and redundant n-tuples, with encouraging results. Although applicable in principle, this technique was not found to be very useful for alphabetic character recognition, mainly because the source data were subject to greater variations (32), so that very few (if any) invariant pattern elements occurred, and similarly there were few redundant n-tuples. Instead, it was decided to adopt more generalised approaches for the research reported here.

2.6 Summary

The n-tuple method of pattern recognition has been introduced and a detailed survey given of previous research, particularly into its use in optical character recognition. Its suitability as a character recognizer for a blind reading aid has been shown, and the areas lacking adequate previous study have been listed, as motivation

for the series of experiments reported in the present thesis. These experiments fall into two general classes: those which explore the capability of n-tuple recognizers to accommodate patterns varying in several of their attributes (such as type, style, positioning, orientation, etc.), and those investigating ways of improving the recognition performance.

The next chapter discusses in detail the experimental techniques employed, particularly in obtaining the source data, setting up the character recognizer structures, and the presentation of the patterns for training and testing.

Chapter 3

Selection of Patterns and Recognizer Structure

3.1 Introduction

This chapter describes in detail the types of pattern likely to be encountered in the proposed reading aid, and the causes of the variations in these patterns which lead to difficulties in recognition. The structure of the simulated recognizer is described, showing how its performance is affected by manipulation of its parameters, and how these should be selected for a particular application. Finally a discussion is presented of the various training, testing, and optimising sets of patterns used in the recognition experiments.

3.2 Pattern Variability

3.2.1 Style and Quality

This section discusses the variations existing in the text material prior to submission to the character recognizer. The most obvious variation to be observed in various examples of text is in the method by which it was produced, viz. handwritten script, typescript, and print, each of which has its own well defined characteristics. Handwriting is notoriously the most difficult form of text to read automatically because it is not standardized in any way and hence has a practically infinite range of variations in style, size, weight, etc. Hand-printed characters are somewhat easier to recognize, especially if attempts at standardization are made, for example in the size of the characters. However, both of these text sources may be dismissed for the present application because the added usefulness of a blind aid capable of reading handwriting would not justify the enormous experimental effort required to produce an adequate recognizer.

Typescript and print are both standardized in that the individual characters are fairly reproducible, and hence within a given piece of text the degree of variation encountered is substantially less than is the case for handwriting. Typescript is simpler than print, in that within a particular style, or font, all characters occupy the same amount of space along the line. Printed text, however, possesses the quality of proportional spacing i.e. different characters cover a range of width, of which 'i' is normally the narrowest, and 'm' the widest. There do exist some forms of typescript which are proportionally spaced. The character recognizer for the proposed reading aid must be capable of handling printed text, and possibly also typescript.

The next form of print variation to be considered is style, or font. There exists a vast range of different fonts, normally divided into several groups by three independent qualities. Firstly the characters may be with or without serifs, which are small appendages attached perpendicularly at the ends of certain limbs of the characters. Secondly, the characters may be old style, or new style. This involves subtle differences in the variation in thickness of the character limbs. Thirdly, the character may be normal (upright),or italic (slanted). By far the majority of bulk printed text is in normal serif print, either old style or new style. When special emphasis is required, a change may be made to an italic or sans-serif font, but these are not generally used for long passages, since they are considered to be less pleasing to the eye and less easy to read.

As well as different fonts, one encounters variations in print density (known as weight, and affecting the character limb thicknesses) and in size, both vertically (point size), and horizontally (compressed

or expanded). Furthermore, the printed impression may not be perfect, in that portions of the character may be missing, or there may be spurious dots of ink around the characters.

Ideally, the character recognizer for a blind reading aid must therefore be able to handle text printed in various fonts, must accommodate imperfections in print quality, and must cope with the more common punctuation marks. Several sets of experiments on the recognition of different fonts are reported in Section 5.2. Both upper-case and lower-case letters are used, in one sans-serif font, two common serif fonts in different weights, and in typescript. The segmentation experiments in Section 5.3 use a common serif font, whilst most of the optimisation experiments in Section 5.4 use typewritten upper-case letters, although most of the results obtained are directly applicable to any type of character.

3.2.2 Random Variations in Characters

This section discusses those variations present in the patterns as they are processed by the pattern classifier which were not present in the source text. The most fundamental form of variation in the patterns is the distortion they undergo when digitized by the input device. A two-level intensity quantization is carried out (see Section 4.2) on the analogue signal representing the pattern. Since the patterns are essentially binary in nature (black or white), there is no significant loss of information at this stage. However, the resolution of the camera is approximately six pixels/millimetre (section 4.2.4) meaning that fine detail in the patterns may be grossly distorted if it is in the region of 1/6 millimetre in size. Furthermore, the exact form of this distortion varies according to the position of the fine detail with respect to the elements of the square photomatrix. The observed image of a character moving across

the field of view constantly changes shape (see Fig.3.1).

A quite straightforward form of pattern variation is the change in position of the character as the field of view scans along the line of text. Both horizontal and vertical alignment of the characters may be influenced in this way, although orientation (rotation) errors should be insignificant. Random noise due to the digitizing electronics may be minimized by averaging several successive frames, although it has been found that if the equipment is not stretched to the limit of its capabilities, this source of noise does not significantly affect the shape of the images. This was shown by using test and training sets of stationary centralized characters which were thus varying only in random noise. Completely correct classifications were consistently achieved.

The next form of pattern variation to be considered is an inescapable consequence of scanning the source text with a fixed size field of view (or 'window'). Firstly, when the window is positioned between two adjacent characters, and part of each is visible, the classifier must ensure that a 'reject' decision is made. Secondly, when reading printed material having proportional character spacing, recognition performance can be degraded either by adjacent characters encroaching on a narrow central character, or by truncation in the case of a wide central character. The means by which these problems are solved is known as segmentation, and attempts to perform this on running text are reported in Section 5.3.

The last important sources of pattern variation are errors in focussing and illumination at the camera. These can drastically affect the input patterns, at low levels by thinning or thickening the character limbs, but at high error levels completely destroying any information in the pattern. It is thus reasonable to expect the recognizer to handle small

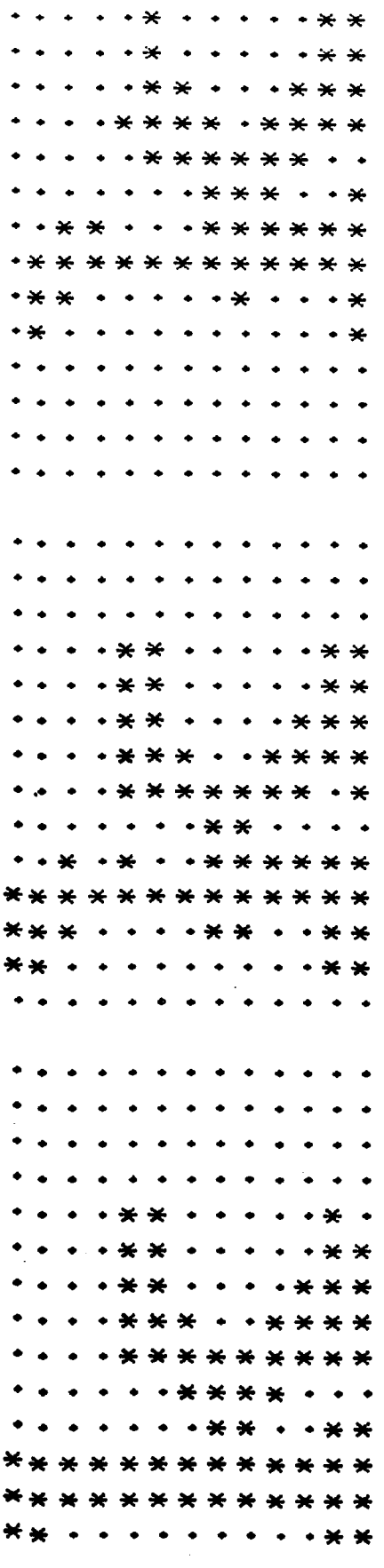


Fig. 3.1 Pattern Distortion.

errors but not gross errors, for which some other means of detection must be sought.

The requirements demanded of the character recognizer to cope with 'non-natural' pattern variations are therefore the following: resolution fine enough not to obliterate important detail; tolerance of errors in character positioning; tolerance of small errors in focussing and illumination; ability to segment running printed text.

3.3 Recognizer Structure

3.3.1 Introduction

Two criteria which may be applied to any pattern recognizer are recognition performance and storage requirements. The n-tuple method has earned a reputation for achieving performance levels comparable with most other general-purpose methods, but usually at the cost of massive storage requirements. In these days of low-cost semiconductor memory such restrictions are becoming less important, particularly when one is able to take advantage of the inherent simplicity and flexibility of the n-tuple method. However, given a recognizer of a particular size, it is important to establish that its storage is being fully utilized. The experiments reported in Sections 5.2 and 5.3 are intended to explore the capabilities of the method in dealing with recognition problems arising in a reading aid for the blind. Section 5.4, however deals with ways of improving the performance within fixed storage constraints, i.e., is concerned with optimisation. It is the purpose of the present section to consider the factors influencing the storage requirements of the recognizer.

The general structure of the recognizer may be established from Figs. 4.2 and 4.3 (in Chapter 4). The patterns appear from the camera or from a computer file as a binary matrix of up to 16 x 16 bits. The

elements of this matrix are mapped arbitrarily onto the n-tuples, i.e. each n-tuple samples a group of n bits selected arbitrarily from the matrix. In hardware terms, the n-tuples are considered as the address input words of random-access memories (RAMs), so that a particular state of an n-tuple addresses a unique location (bit) within the appropriate memory. A discriminator is a group of n-tuples (or RAMs) which is trained on patterns of one recognition class only, and all the n-tuple outputs (RAM data outputs) are summed to give the discriminator output, or response. As outlined in section 2.2, the training process involves showing the recognizer a series of patterns of known class, and within the discriminator appropriate to the class of a particular pattern setting the location addressed in each n-tuple store (or RAM) by the state of its input word. Each class has a discriminator trained in this way. The testing process involves showing the recognizer patterns of unknown class, this time obtaining the response of every discriminator to a particular pattern, and making the decision 'class q' if the qth discriminator has the largest response.

The factors affecting storage are n-tuple size, pattern dimensions, number of n-tuples, and number of discriminators. Two further structural aspects considered below are the mapping, and the decision logic.

3.3.2 n-tuple Size

The influence of n-tuple size on the storage requirements is quite simple, since the number of memory locations needed to accommodate all the possible states of an n-tuple is simply 2^n. If the pattern matrix has R elements then the number of n-tuples needed for a complete 1:1 coverage is k, where $k = \frac{R}{n}$

and is assumed for simplicity to be an integer. Thus the number of storage locations needed for one discriminator is S, where

$$S = k.2^n = \frac{R}{n} . 2^n$$

If there are C recognition classes, and hence C discriminators, the overall storage requirements are T locations, where

$$T = \frac{CR}{n} . 2^n$$

since all the discriminators are structurally identical. In all the recognition experiments reported here, each storage location is simply one bit. The mapping is not limited in every case to 1:1, in particular for the experiments reported in section 5.4.1.

The relationship between n-tuple size and recognition performance is more complex, since it is a fundamental parameter of the recognizer structure. In general, larger n-tuples yield better recognition performance, but require larger training sets to achieve it. There are two reasons for this. Firstly a larger training set is normally more fully representative of the set of possible test patterns and hence can lead to better discriminant functions. Secondly, the decision surfaces employed by a large n-tuple system are of a higher order of complexity than those of a smaller n-tuple system. Therefore the large n-tuple system is sensitive to smaller differences between patterns of different classes, and thus, after sufficient training, can classify more test patterns correctly. The learning curves in Fig.3.2 for typewritten letters clearly show the improvement in performance as n increases, confirming Ullmann's findings (22). Note that the performance peaks occur at different training set sizes, and that for the smaller systems (n = 4 and n = 6) further training causes a decrease in performance, which is not the case for n= 8. One would normally expect learning curves to exhibit a plateau (i.e., a levelling-out of performance) when as much information as possible has been absorbed from the training patterns

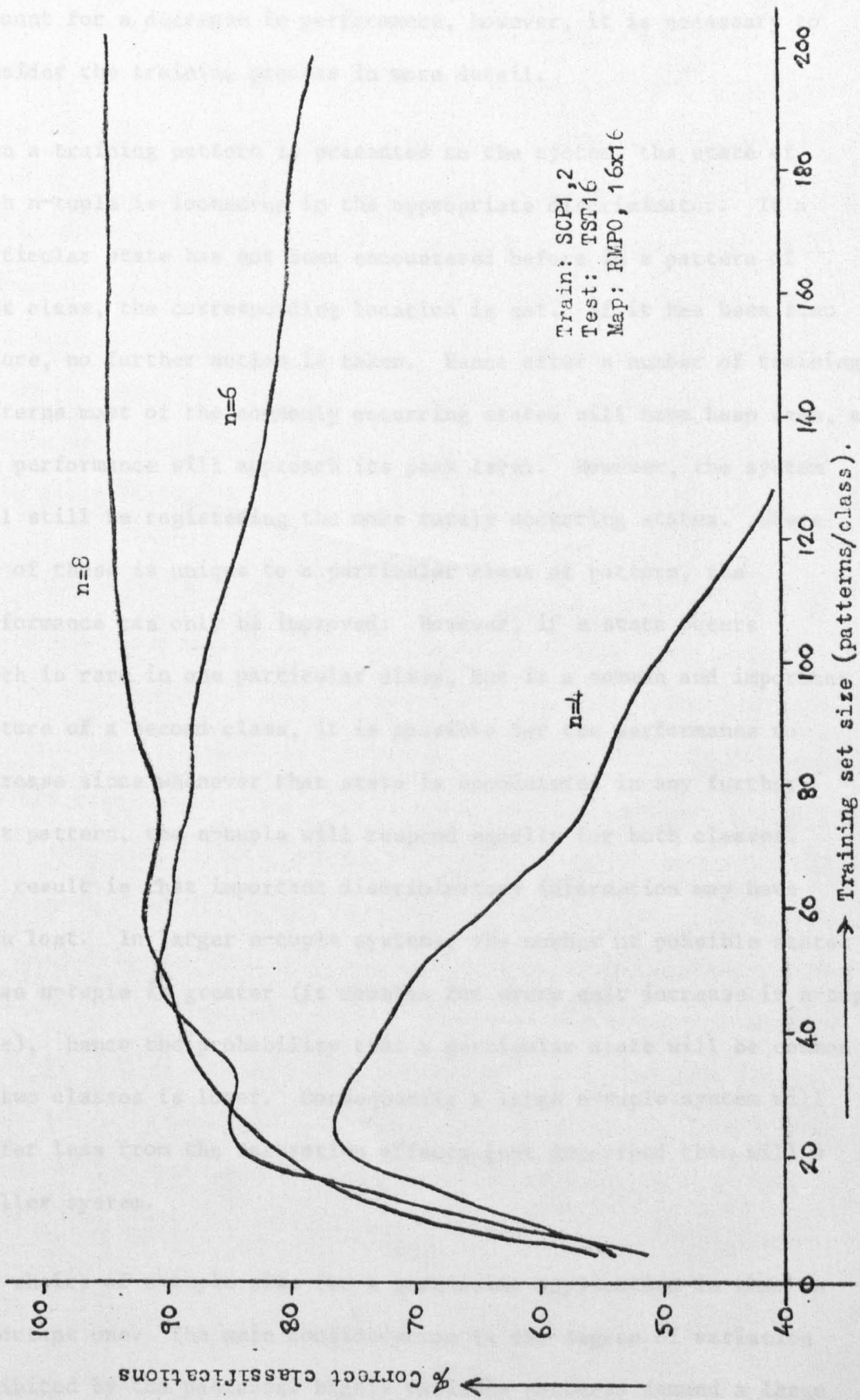


Fig. 3.2 Learning Curves for n=4,6,8.

(the limitation being the complexity of the system). In order to account for a decrease in performance, however, it is necessary to consider the training process in more detail.

When a training pattern is presented to the system, the state of each n-tuple is looked-up in the appropriate discriminator. If a particular state has not been encountered before in a pattern of that class, the corresponding location is set. If it has been seen before, no further action is taken. Hence after a number of training patterns most of the commonly occurring states will have been seen, and the performance will approach its peak level. However, the system will still be registering the more rarely occurring states. Where one of these is unique to a particular class of pattern, the performance can only be improved. However, if a state occurs which is rare in one particular class, but is a common and important feature of a second class, it is possible for the performance to decrease since whenever that state is encountered in any further test pattern, the n-tuple will respond equally for both classes. The result is that important discriminatory information may have been lost. In larger n-tuple systems, the number of possible states of an n-tuple is greater (it doubles for every unit increase in n-tuple size), hence the probability that a particular state will be common to two classes is lower. Consequently a large n-tuple system will suffer less from the saturation effects just described than will a smaller system.

The choice of n-tuple size for a particular application is thus an important one. The main consideration is the degree of variation exhibited by the patterns; highly variable patterns demand a large n-tuple size in order to minimize saturation. Conversely, relatively stable patterns may be adequately handled by a smaller system.

However, in general one should aim to use as large an n-tuple size as practicable, provided that care is taken to ensure that the training set is sufficiently large. The safest way to do this is to plot a learning curve for the system, i.e., a graph of performance against training set size. Most of the experiments reported in Chapter 5 use an n-tuple size of 8, which was the largest that could be usefully employed in the computer simulations. In practical terms, there would be definite advantages in using larger n-tuple sizes, e.g. n = 10 means that the memory elements would each contain 1k bits - a very convenient size. Further discussion of implementation may be found in Chapter 6.

3.3.3 Pattern Dimensions

Ideally the input matrix of the character recognizer being considered would enclose exactly a single character of the text, with a minimum of 'unused' space around the character. The number of pattern elements which are used to represent the character obviously defines the resolution of the input system: more pattern elements give finer resolution and less distortion of the image, but obviously at the cost of more n-tuples to provide adequate coverage. For essentially graphical patterns such as letters of the alphabet a useful guide to the necessary resolution may be obtained by simple visual inspection of the digitized images: if they are not recognizable the resolution is obviously insufficient. For most of the experiments reported here a matrix of 12 x 16 bits was employed, and the resolution adjusted, by altering the distance between the text and the camera, so that a typical character just filled the matrix. Distortion of the characters due to resolution effects was found to be negligible except in the case of some small lower-case printed letters such as 't' and 'f' which were occasionally rendered indistinguishable. These cases are given special mention in section 5.2.

3.3.4 Number of n-tuples

Having fixed the n-tuple size and pattern dimensions it is necessary to establish the number of n-tuples to be used. The most widely used configuration involves a simple 1:1 mapping between the pattern elements and the n-tuple bits, i.e., each pattern element is covered once only, and each n-tuple bit is connected to only one pattern element. However, the recognition performance does improve as the number of n-tuples is increased, i.e., the mapping becomes 1:many. A set of experiments concerning the variation in performance with the number of n-tuples is reported in section 5.4. In some cases it was found that doubling the number of n-tuples of a given size was comparable in effect to increasing the n-tuple size by one whilst keeping the number of n-tuples constant. In both cases the increase in storage space was the same. Thus for aparticular application the choice of the number of n-tuples can be as important as the choice of n-tuple size.

3.3.5 Number of Discriminators

All of the recognition experiments reported here employ exactly one discriminator per pattern class, thus for a repertoire of 26 classes, to cover the alphabet, 26 discriminators are required. Where recognition of both upper and lower-case alphabets is desired one has the option of either using 52 discriminators, training each on only one type of pattern, or using 26 discriminators, training each on examples of both upper and lower-case letters. Experiments on these two strategies are described in section 5.2. In general, extension of the number of discriminators or of the range of the training patterns inevitably degrades the overall recognition performance of the system. Some of the experiments on optimisation, reported in section 5.4,used only five discriminators, for the classes C, G, O, Q and U, which was one group of letters causing a large number of confusions. Restriction

to only five classes reduced the running time for the computer simulations so that the experiments could be taken several stages further in the time available than would have been the case for 26 classes.

No attempts were made to handle punctuation, though this would be desirable in a reading aid. No special difficulties, however, are anticipated in extending the recognizer to cope with the more common punctuation marks, but how the utterance generator would handle such features is felt to lie outside the scope of the present thesis.

3.3.6 Mappings

Most of the recognition experiments are based on mappings built up using a pseudo-random number generator. All of the complete 1:1 mappings possess the permutation property discussed by Prorok (34) i.e., every pattern element is connected to exactly one n-tuple bit, and every n-tuple bit is connected to just one pattern element. However, some of the segmentation experiments (section 5.3), and some optimisation experiments (section 5.4) use 1:many mappings in which each pattern element may be connected to as many as four n-tuple bits (belonging to different n-tuples), still in an essentially random manner.
Other optimisation experiments use non-random maps built up according to various sets of rules, whilst still further experiments use a number of different pseudo-random maps simultaneously, e.g. six mappings may be shared in an arbitrary manner by the discriminators.

3.3.7 Decision Logic

Given the set of discriminator responses to a particular test pattern the purpose of the decision logic is to establish whether the pattern is to be rejected, or if not, to which class it belongs. In the simplest case, one may reject all those patterns which cause two

or more discriminators to give an equal maximum response (known as a 'tie'), and in the case of a unique maximum response accept the class of the corresponding discriminator as that to which the pattern belongs. This choice of decision logic gives the highest number of correct classifications, but also the highest number of mis-classifications. In other words, any threshold system is bound to increase the number of rejections, simply by discarding low-confidence decisions as reflected by the response sizes. The only exception is where the recognizer is to be presented with unrecognizable patterns, i.e., ones which do not belong to any of the classes on which the recognizer has been trained. Rejection of such a pattern would be counted as a correct decision.

Thresholds may be implemented in several ways, namely absolute, relative, and combined absolute and relative. In an absolute threshold system, a pattern is rejected if there is an ambiguous response (two or more discriminators giving an equal maximum response) or if the unique maximum response lies below the present threshold value. Fig.3.3(b) shows the effect of increasing the absolute threshold value on the numbers of correct and incorrect classifications for a typical set of results. Note that below the value 12 the threshold has no effect, and that thereafter the number of misclassifications falls much more rapidly than the number of correct classifications. This is the ideal behaviour, since all the errors can be eliminated (at threshold = 20) without much loss of performance. A relative threshold system is implemented by rejecting those patterns which produce a maximum response which differs from the next highest response by less than the preset threshold value. This obviously includes the case of ambiguity, where the difference value is zero. Fig.3.3(a) shows the effect of increasing relative threshold values for the same set of responses as previously. Above a relative

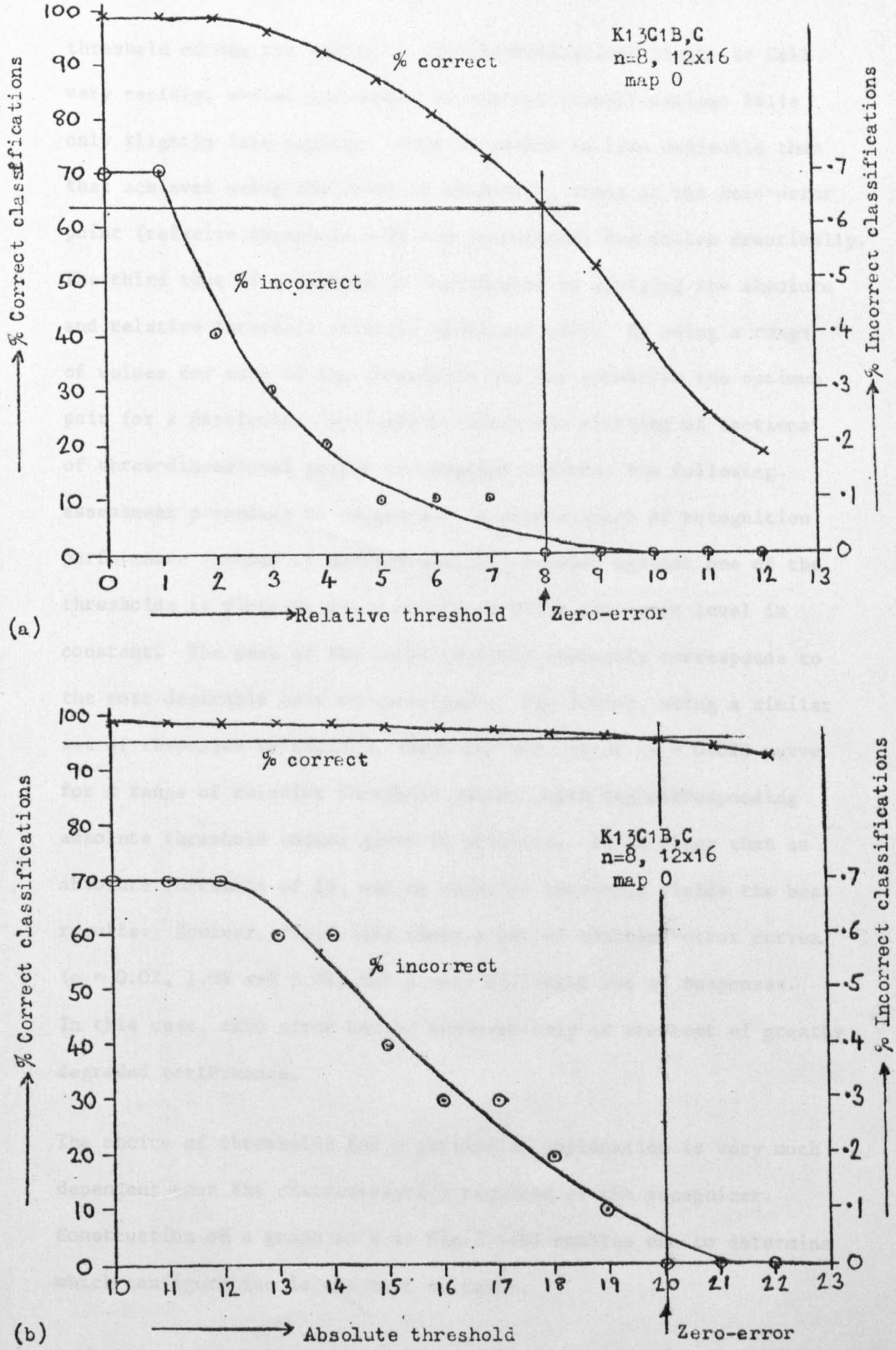
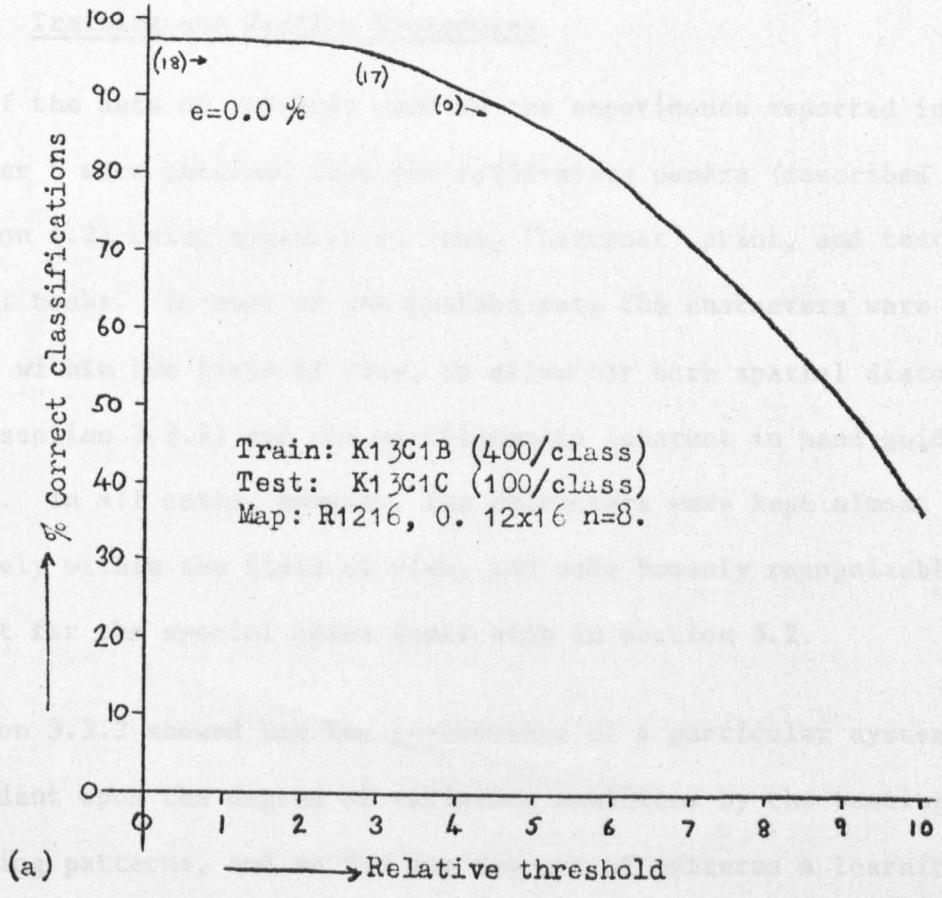


Fig. 3.3 Relative and Absolute Thresholds.

threshold of one the number of mis-classifications starts to fall very rapidly, whilst the number of correct classifications falls only slightly less rapidly. This situation is less desirable than that achieved using the absolute threshold, since at the zero-error point (relative threshold = 8) the performance has fallen drastically. The third type of threshold is implemented by applying the absolute and relative threshold criteria simultaneously. By using a range of values for each of the thresholds one may establish the optimum pair for a particular application. Since the plotting of sections of three-dimensional graphs is somewhat tedious, the following assessment procedure is suggested. A simple graph of recognition performance (number of correct classifications) against one of the thresholds is plotted, using points at which the error level is constant. The peak of the curve obtained obviously corresponds to the most desirable pair of thresholds. Fig.3.4(a), using a similar set of responses to Fig.3.3, shows the zero-error (e = 0.0%) curve for a range of relative threshold values, with the corresponding absolute threshold values given in brackets. It is clear that an absolute threshold of 18, and no relative threshold yields the best results. However, Fig.3.4(b) shows a set of constant-error curves (e = 0.0%, 1.0% and 5.0%) for a very different set of responses. In this case, zero error can be achieved only at the cost of greatly degraded performance.

The choice of thresholds for a particular application is very much dependent upon the characteristics required of the recognizer. Construction of a graph such as Fig.3.4(b) enables one to determine which configuration is the most suitable.



(e=error rate; figures in brackets are absolute threshold values)

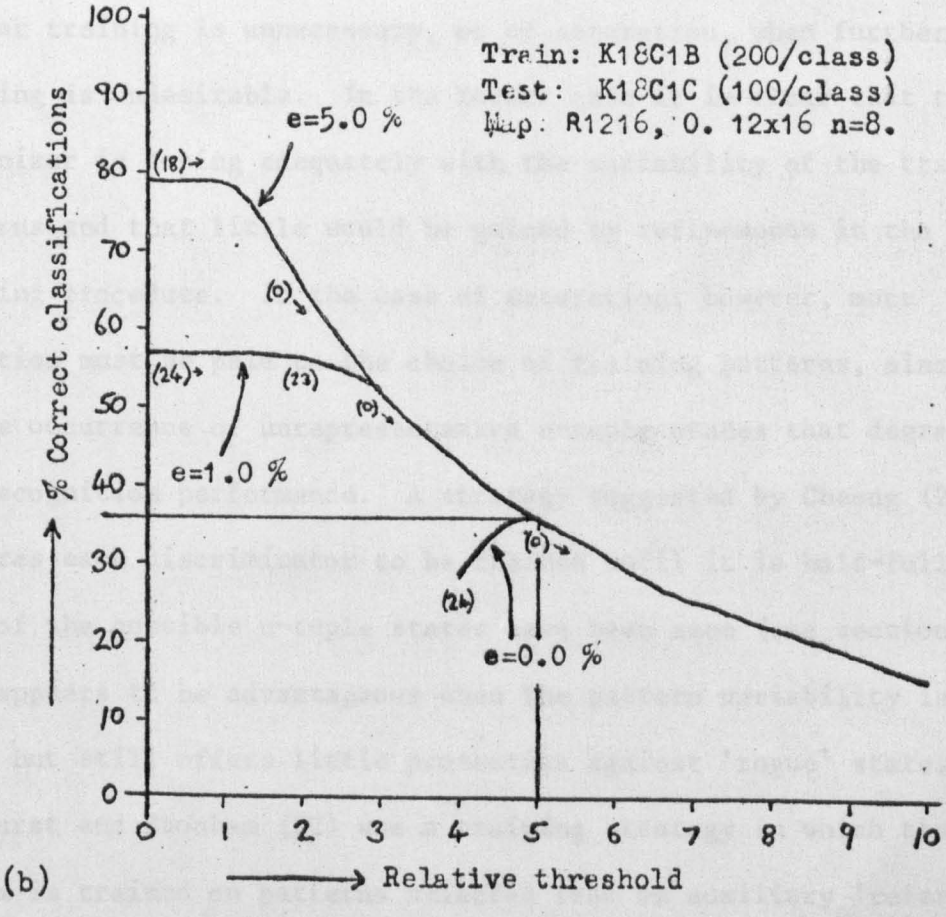


Fig. 3.4 Combined Absolute and Relative Thresholds.

3.4 Training and Testing Procedures

All of the sets of patterns used in the experiments reported in Chapter 5 were obtained from the solid-state camera (described in section 4.2) using typewritten text, 'Letraset' print, and text from normal books. In most of the pattern sets the characters were moved about within the field of view, to allow for both spatial distortion (see section 3.2.2) and the misalignments inherent in hand-guided input. In all cases, however, the characters were kept almost entirely within the field of view, and were humanly recognizable, except for the special cases dealt with in section 5.2.

Section 3.3.2 showed how the performance of a particular system is dependent upon the degree of variation exhibited by the testing and training patterns, and so for any new set of patterns a learning curve of recognition performance against training set size is plotted. This clearly shows the onset of the plateau region, when further training is unnecessary, or of saturation, when further training is undesirable. In the former case it is clear that the recognizer is coping adequately with the variability of the training patterns and that little would be gained by refinements in the training procedure. In the case of saturation, however, more attention must be paid to the choice of training patterns, since it is the occurrence of unrepresentative n-tuple states that degrades the recognition performance. A strategy suggested by Cheung (28) requires each discriminator to be trained until it is half-full, i.e. half of the possible n-tuple states have been seen (see section 2.2). This appears to be advantageous when the pattern variability is high, but still offers little protection against 'rogue' states. Fairhurst and Stonham (32) use a training strategy in which the system is trained on patterns selected from an auxiliary 'reference' set if the recognizer is unable to classify them correctly. This

helps to keep the training set small, and consequently delays the onset of saturation, but nevertheless 'rogue' states are still included. A series of experiments reported in section 5.4 makes use of the occurrence frequencies of n-tuple states, without needing to store them in the recognizer memory locations, in an effort to place more weight on the more frequently occurring states.

In all optimisation experiments involving the adjustment of the recognizer in the light of its performance on a set of patterns it is necessary to reserve a completely separate set of patterns which is used solely for assessing the recognition performance. The need for an independent test set is seen when one considers that the best recognition performance could be achieved simply by including the test set within the training set. Most of the optimisation experiments reported here make unrestricted use of a separate optimisation set, leaving the test set for independent measurements of performance.

Some of the segmentation experiments presented in section 5.3 call for the use of a set of 'reject-class' patterns, i.e. of patterns of indeterminate class which should be rejected. For example, when two halves of printed letters are visible on opposite sides of the scanning window the resultant pattern must be rejected. Such patterns are labelled as 'class zero', and all rejections are considered to be class zero decisions. Thus rejection of a class zero pattern is counted as a correct decision, and may contribute to the recognition performance. It is not desirable to train the recognizer on class zero patterns, however, because their inherently large degree of variability means that the class zero discriminator would be heavily saturated, and would respond strongly to very many test patterns.

3.5 Summary

In this chapter the nature of the patterns to be recognized has been detailed, with a consideration of the variations which they may undergo both as a result of variations in the printed characters, and as a result of processing within the recognizer. The structure of the character recognizer was described in detail, with particular emphasis on the more important parameters, namely n-tuple size, pattern dimensions, number of n-tuples, number of discriminators, mappings, and the decision logic. Finally was a discussion of the various sets of patterns employed for training, testing, and optimising the recognizer. Chapter 4 presents descriptions of the hardware systems: camera, interface, and computer, and the software subsystems for handling the patterns and simulating the recognizers.

Chapter 4

Hardware and Software Systems

4.1 Introduction

In the preceding chapters, references were made both to a solid-state camera system giving digitized images of text for recognition experiments, and to a series of programs for handling these images (patterns), simulating various structures of pattern classifier, and analysing the performance of these structures. In this chapter these two systems, the hardware and the software, are described in greater detail. Reference should also be made to Appendix A for circuit diagrams of the camera system, and to Appendix B for a guide to the use of the various programs.

4.2 The Camera System

In order to undertake the character recognition experiments, it was necessary to provide a supply of real data obtained by a camera from appropriate printed text. For a number of reasons, including simplicity of design and ease of implementation, it was decided to use a 64 x 64 bit self-scanned photodiode array mounted in a compact camera head. A conventional camera lens was used, with a close-up bellows unit to permit variable resolution. Supplied with the integrated circuit array was a video processor board giving a shaped analogue video output, though it was necessary to develop further circuitry, falling into three function groups (see Fig.4.1).

4.2.1 Camera driving logic (Figs.A.1, A.2 (Appendix A))

This includes a variable frequency clock which determines the rate at which the diode matrix is scanned, and hence the overall light sensitivity of the camera (because the amount of recharge necessary for each photodiode is dependent upon the total charge leakage since it was last scanned, and this in turn is dependent upon the light

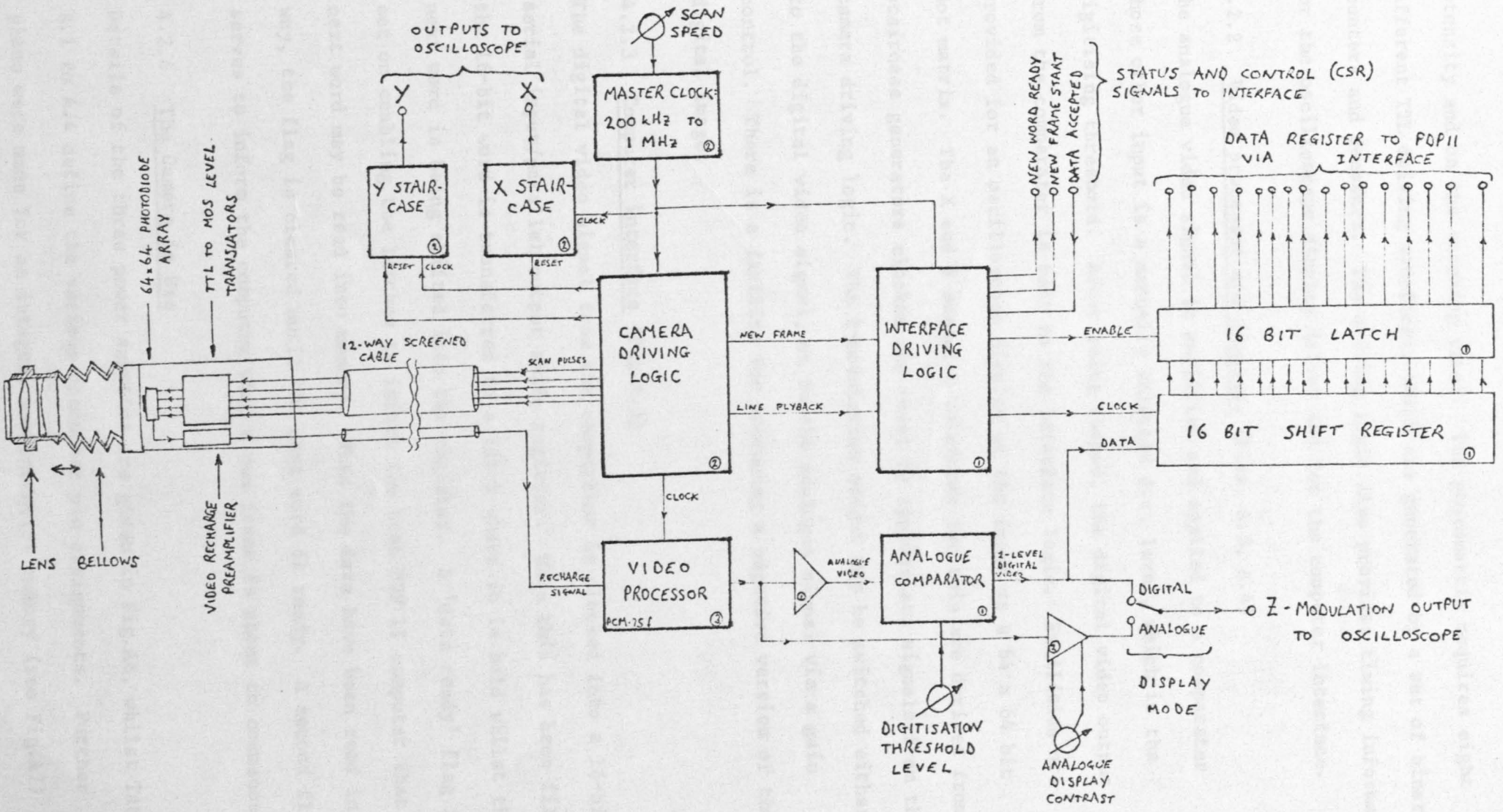


Fig. 4.1 Solid-state Camera System.

intensity and on the scanning rate). The photomatrix requires eight different TTL driving waveforms, which are generated by a set of binary counters and decoders. The driving logic also provides timing information for the oscilloscope display driver and for the computer interface.

4.2.2 Video processor and display. (Figs. A.3, A.4)

The analogue video signal is amplified and applied to a comparator whose other input is a manually selected d.c. level, which is the digitising threshold. After being shaped, the digital video output from the comparator is sent to the interface logic. Facilities are provided for an oscilloscope display of the image as a 64 x 64 bit dot matrix. The X and Y scanning waveforms for this are derived from staircase generators clocked and reset by appropriate signals from the camera driving logic. The Z-modulation output may be switched either to the digital video signal, or to the analogue signal via a gain control. There is a facility for producing a negative version of the digital image.

4.2.3 Computer interface (Fig.A.5)

The digital video signal from the comparator is clocked into a 16-bit serial input/parallel output shift register. When this has been filled, the 16-bit word is transferred to a latch where it is held whilst the next word is being shifted into the register. A 'data ready' flag is set on enabling the latches to inform the host PDP 11 computer that the next word may be read into memory. When the data have been read in this way, the flag is cleared until the next word is ready. A second flag serves to inform the computer when a new frame is about to commence.

4.2.4 The Camera in Use

Details of the three power supplies are given in Fig.A6, whilst Tables A.1 to A.4 define the various connecter pin assignments. Further plans were made for an integral semiconductor memory (see Fig.A7)

capable of accumulating up to sixteen successive frames from the camera, and then sending the resultant average picture to the computer. The advantage of such an arrangement is that it can remove a large proportion of the noise present in the two-level digital image, particularly at the edges of the character. A more detailed discussion of the characteristics of this noise may be found in Section 3.2. Reference (36) describes the noise reducer design in detail, though construction was not completed.

In use, the camera was mounted on a tripod and pointed vertically down onto the object being digitized, which could be moved around on the top of a table to enable a line of text to be 'scanned'. Illumination was provided by a conventional 60W tungsten lamp on each side, at about 15 cm from the object. A typical procedure would be to digitize and store on a computer file many patterns of each class whilst displacing the object slightly in order to simulate tracking errors. The stored patterns would later be recovered for use as training or testing sets. In practice negative digital images were used, i.e. set bits corresponded to the dark, printed areas of the original object. The camera resolution was normally set (by means of the bellows) such that a typical printed letter approximately 2.0 mm wide and 2.5 mm high was contained within a 12 x 16 bit window extracted from the original 64 x 64 bit matrix by the computer. The resolution was thus approximately 6 bits per mm.

The density of the digitized image is dependent upon several factors, viz. digitizer threshold setting,clock speed, illumination intensity, object contrast, and camera lens aperture setting. The best performance was achieved with a large aperture, high light intensity and a fast clock speed. The threshold was adjusted to suit the object contrast. The advantage of the high scanning rate is that

it minimizes charge leakage in unilluminated photodiodes, thus reducing image 'spread'. A discussion of the effects of the various photodiode characteristics on the image may be found in reference (37).

4.3 Host Computer System

The computer on which all the pattern recognition experiments were performed, and to which the camera was interfaced, was a DEC PDP 11/40 having 28k words of core memory, extended instruction set, twin disk cartridge drivers, dectape drivers, magnetic tape drive, various terminals,line printer, and a VT11 graphics display. The latter was used to great advantage in displaying both 64x64 bit and 16x16 bit images, as well as providing a fast and efficient means of interacting with the various recognition and analysis programs.

4.4 The Software System

4.4.1 Introduction

Several programs were written, all running under the RT-11 operating system on a PDP 11/40 computer, and falling into four groups:

(i) Pattern manipulation: EDIPIC, written in MACRO-11 assembly language.

(ii) Pattern classification: MULTI (developed from an earlier program called RECIPE) and MULTI2, both written in MACRO.

(iii) Response analysis: ANGELA, written in FORTRAN.

(iv) Optimisation: MAPPIT, HAMAND, FOSOC (written in MACRO) and FRAN (written in FORTRAN).

All the programs written in MACRO-11 make use of a common run-time subsystem, known as JANSYS. This provides all the input/output

handling, support of the graphics terminal, handling of pattern files, and miscellaneous other services requested by the master program. In addition, JANSYS provides an interpreter for the program command strings, which are written in a language designed in such a way that single elementary operations can be executed one at a time, or alternatively concatenated into complex processing sequences. Programs written in this way may be stored permanently as files, to be executed on request. A users' guide to JANSYS may be found in Appendix B.

There follows a description of each of the programs mentioned above.

4.4.2 Pattern Manipulation: EDIPIC

This program contains all the modules required for preparing files of patterns for later use as training or testing sets. It is capable of receiving and displaying 64x64 bit pictures from the camera, extracting a movable 'window' of variable dimensions from this picture, and performing other functions such as the averaging of several frames in order to reduce the effects of random noise. It further allows one to create or amend patterns by selectively setting or clearing their constituent bits. Both 64x64 and 16x16 patterns may be converted into legible form and printed. The users' guide to EDIPIC (Appendix B) makes reference to map files having the extension 'RMP'* (see footnote). These map files are suitable for use only with an old form of the pattern classifier: RECIPE. The current version, MULTI, uses map files having the extension 'MMP', which are created exclusively by the program MAPPIT, described below.

---

* The term 'extension' refers to that part of the file name which defines its nature, e.g., 'RMP' means Random MaP', 'TXT' means 'TEXT', etc. Further clarification may be obtained from (38).

4.4.3 Pattern Classification

(i) MULTI. This program allows one to build an n-tuple pattern classifier suited to the task in hand by control of several attributes, namely, pattern dimensions, n-tuple size, connection mapping between patterns elements and n-tuples, and the number of pattern classes, the names of which may be freely chosen.

The pattern x and y dimensions may each be set to any value in the range 1 to 16, the configuration used in the majority of experiments being 12 x 16.

The n-tuple size, i.e. the number of pattern elements in each n-tuple, may be set at any value from 4 to 8. It should be noted that the storage space needed is an exponential function of the n-tuple size.

The connection mapping is very flexible, in that it allows any number of n-tuples to be connected to the input pattern in any way desired, provided that each constituent bit of an n-tuple is connected to only one pattern element. There is no restriction on the number of n-tuples to which a particular pattern element is connected, although storage limitations must always be considered. MULTI permits only one mapping to be in effect at any time, so that all classes must share the same mapping.

The classifier may have up to 32 discriminators (see Fig.4.2) so that the 26 classes required for recognition of letters are easily supported. Appendix B gives details of the individual commands used in operating the classifier, but a brief description of the processes involved is given below.

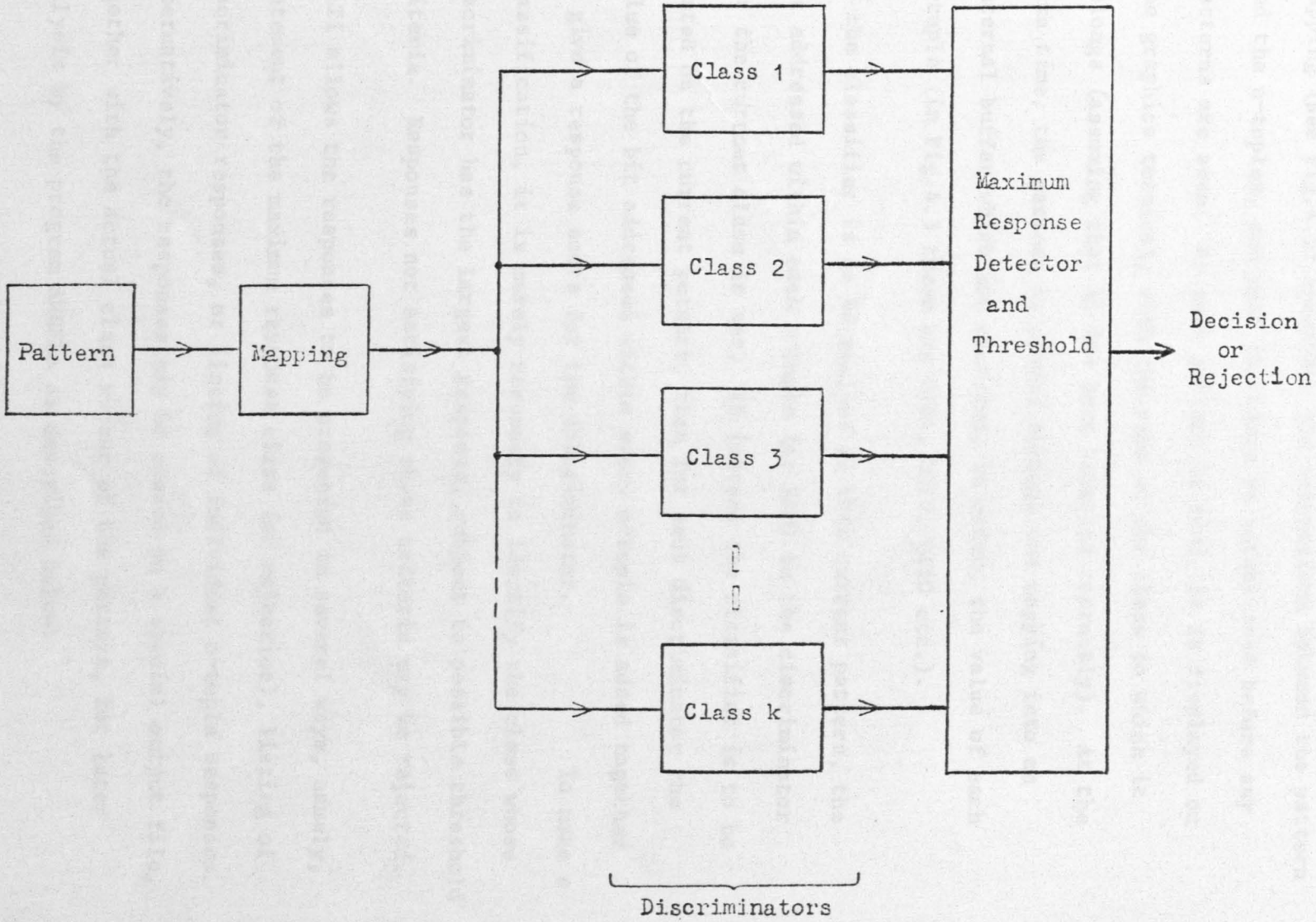


Fig. 4.2 Single Map Pattern Classifier: MULTI.

The classifier normally possesses three input channels which are used for commands, patterns (of up to 16x16 bits), and maps. The mapping (see Fig.4.3) specifies the connections between the pattern and the n-tuples, and must therefore be established before any patterns are seen. As each pattern is read, it is displayed on the graphics terminal, with the name of the class to which it belongs (assuming that it has been labelled correctly). At the same time, the pattern is passed through the mapping into an internal buffer which now contains, in order, the value of each n-tuple (in Fig.4.3 these are 0101, 0010, 0110 etc.).

If the classifier is to be trained on this current pattern, the bit addressed within each n-tuple (or RAM) in the discriminator for the current class is set. If instead the classifier is to be tested on the current pattern, then for each discriminator the value of the bit addressed within every n-tuple is added together to give a response score for the discriminator. To make a classification, it is merely necessary to identify the class whose discriminator has the largest response, subject to possible threshold criteria. Responses not satisfying these criteria may be rejected.

MULTI allows the responses to be presented in several ways, namely, statement of the maximum response class (or rejection), listing of discriminator responses, or listing of individual n-tuple responses. Alternatively, the responses may be stored on a special output file, together with the actual class number of the pattern, for later analysis by the program ANGELA as described below.

Further facilities offered by MULTI include the use of a manually adjustable testing threshold which may be absolute (i.e. if the largest response is less than the threshold, the pattern is rejected) or relative (i.e. rejection occurs if the difference between the two

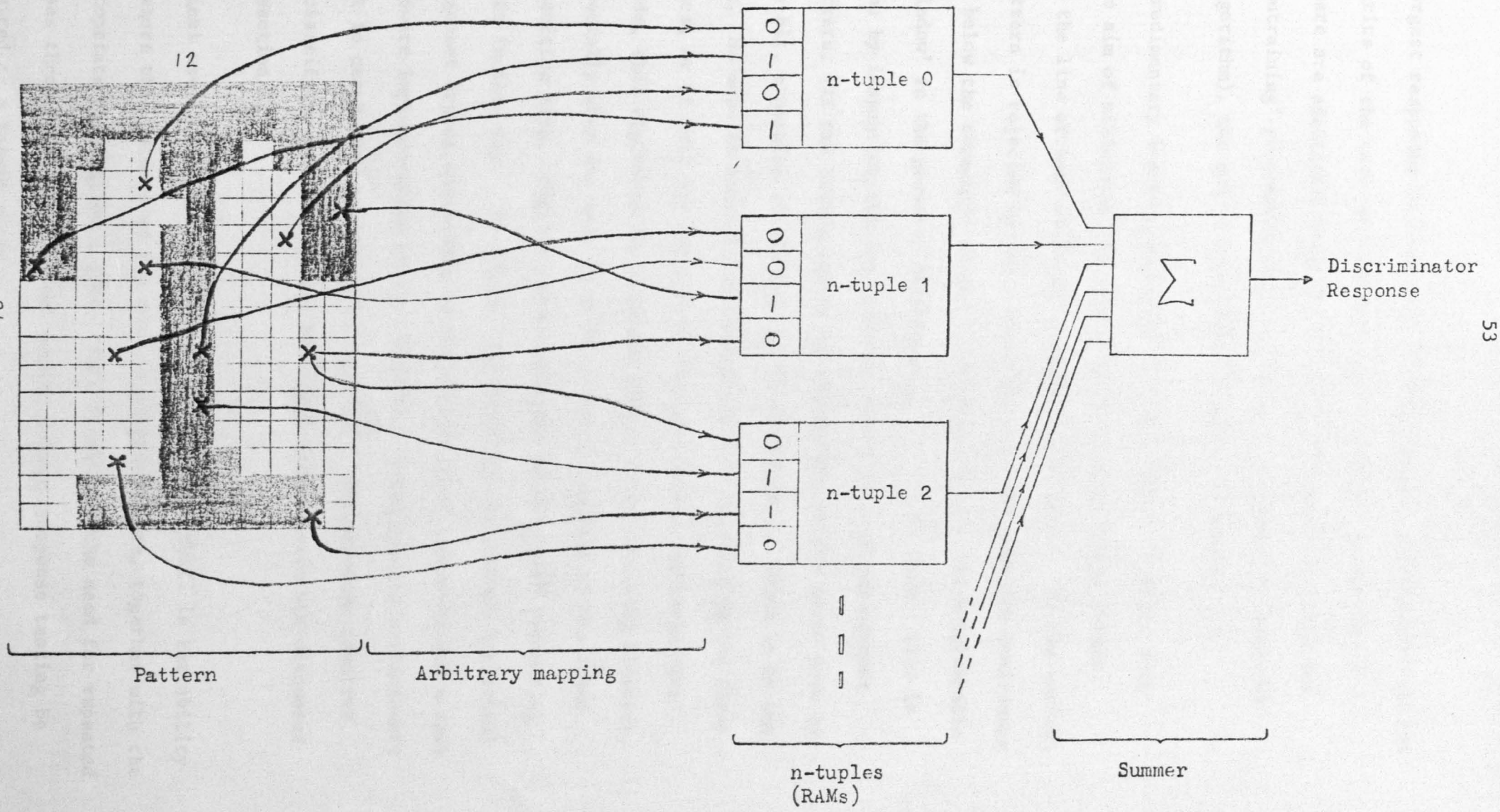
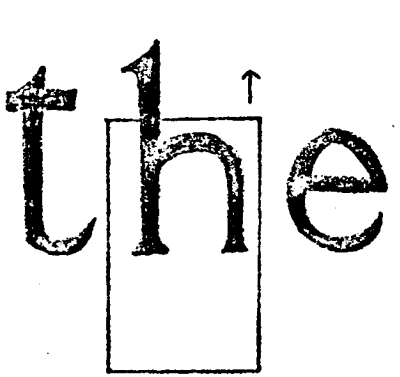


Fig. 4.3 Discriminator Structure.

largest responses is less than the threshold). A discussion of the merits of the various threshold systems appears in section 3.3.7. There are additional commands to permit selective training and 'untraining' procedures (e.g. a 'train if response is incorrect' algorithm), the use of which was discussed in section 3.4.

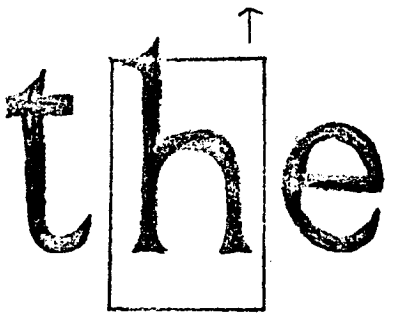
A rudimentary tracking algorithm is incorporated in MULTI with the aim of minimizing the effect of vertical tracking errors as the line of text is being scanned by the camera. If the current pattern is rejected by the classifier, i.e. the decision confidence is below the threshold value, it attempts to move the programmable 'window' in the probable direction of the line of text. This is done by inspecting the top and the bottom rows of the current pattern. If the bottom row is not blank, the window moves down by one bit, otherwise if the top row is not blank, it moves up by one bit. No move is made if the pattern was not rejected in the first place, or if both top and bottom rows are blank. Although very crude, this algorithm has a marked effect on the tracking accuracy, especially when the rate of drift is small compared to the frame repetition rate. Fig.4.4 illustrates the principle of correcting drift in this way. No attempt has been made to correct horizontal alignment errors, since this is the direction of scanning, and a real hardware implementation of the classifier would operate sufficiently fast to catch the characters 'on the fly'. The problems involved in classifying patterns from a hand-held camera probe are discussed in section 3.2.

A final useful feature of the classifier program MULTI is the ability to store the contents of the trained discriminators, together with the appropriate mapping, on a file. This eliminates the need for repeated passes through the training set should further response testing be required at a future stage.



MAX. RESPONSE: ?

Rejection: move window UP



MAX. RESPONSE: ?

Rejection: move window UP

MAX. RESPONSE: h

Decision: no move

Fig. 4.4 Window Tracking Action.

(ii) MULTI2: This is an extended version of MULTI, and offers all of the facilities described above. In addition, it is capable of supporting a number of independent maps simultaneously, as shown in Fig. 4.5. Each map serves at least one discriminator in an arbitrary manner selected during the initial dialogue phase of MULTI2, although of course each discriminator can be served by only one map. MULTI2 was developed in order to test ideas on the optimisation of mappings, of which a discussion may be found in section 5.4.

4.4.4 Response Analysis ANGELA

The response files produced by pattern classification programs such as MULTI contain as many fixed length records as there were patterns in the testing set. Each record contains the response of each discriminator to the pattern, together with the actual class number of the pattern. At the beginning of the file is a header containing a descriptive title, with such details of the classifier as pattern dimensions, n-tuple size, threshold, etc. In processing the responses, ANGELA first establishes the maximum response, according to the absolute or relative threshold rules (see i) above), then records the decision in a set of running totals for the whole file. When all the patterns have been processed in this way, a pair of confusion matrices is printed, the first showing where all the unique classifications occurred (i.e. which discriminator responded to a pattern of a particular class), together with rejections due to threshold effects. The second confusion matrix lists all of the ambiguous responses, where more than one disciminator shared the same maximum score for a particular pattern. Fig.4.6 is an example of the first confusion matrix,

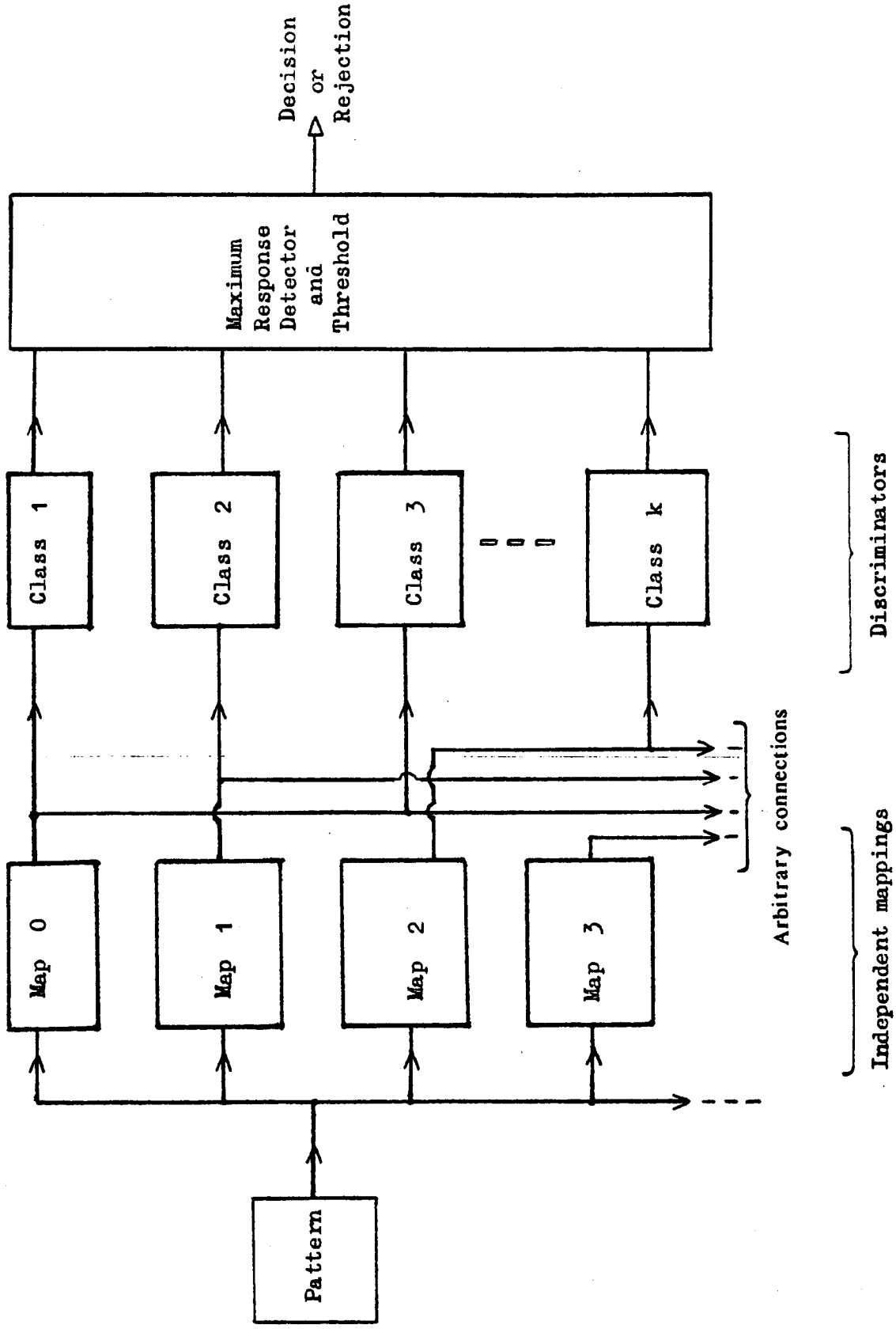


Fig. 4.5 Multiple Map Pattern Classifier: MULTI2.

Train: N12A1A.C16 Test: N12A2B.C16 Map 0. t=3 rel.
Records seen : 2600. Elements used : 10503. Average record size : 4.0
Created : 3-JUL-77 N-tuple size : 8 Window dimensions : 12x16
Test threshold overridden during analysis only. Original value : 3

Pattern Class Counts

	?	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
0	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

Relative thresh. Correct Incorrect Rejected

2 91.8 % 0.7 % 7.5 %

Unique Classifications

Pattern Class

	?	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
?	.	.	.	12	1	.	.	6	.	.	.	3	11	.	3	31	19	38	33	36	1	.	.
A	.	100	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
B	.	.	100	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
C	.	.	.	87	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
D	.	.	.	.	99	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	100	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	100	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
G	.	.	.	.	.	.	.	94	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
H	.	.	.	.	.	.	.	.	100	.	.	.	.	.	.	.	.	.	1	.	.	.	.
I	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.	.	1	.	.	.	1	.	.
J	.	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.	.	.	.	.	.	.	.
K	.	.	.	.	.	.	.	.	.	.	.	97	.	.	.	.	.	.	.	.	.	.	.
L	.	.	.	.	.	.	.	.	.	.	.	.	89	.	.	.	.	.	.	.	.	.	.
M	.	.	.	.	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.	.	.	.	.
N	.	.	.	.	.	.	.	.	.	.	.	.	.	.	97	.	.	.	.	.	.	.	.
O	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	67	.	2	.	.	.	.	.
P	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	79	.	.	.	.	.	.
Q	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	60	.	.	.	.	.
R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	66	1	.	.	.
S	.									.	.	.	.	.	.	.	.	.	.	55	.	.	.
T	.									.	.	.	.	.	.	.	.	.	.	.	98	.	.
U	.									.	.	.	.	.	.	2	.	.	.	6	.	100	.

Fig. 4.6 Confusion Matrix (in part).

showing all of the unique classifications and threshold rejections for 100 patterns each of the classes A to Z. Patterns classified as '?' are rejected because their responses are less than the currently set threshold (which has the value 2, relative, in this case). For example, 66 patterns of class 'R' were correctly classified, 1 pattern was misclassified as an 'H', and 33 were rejected.

4.4.5 Optimisation

(i) MAPPIT. This program handles the mappings needed by the classification programs MULTI and MULTI2. The mapping determines the way in which elements of the pattern are connected to the n-tuples (i.e. the RAM address inputs), as illustrated in Fig.4.3. The prime function of MAPPIT is to generate a random (or pseudo-random) mapping for a particular classifier having specified characteristics, namely pattern dimensions, n-tuple size, and number of n-tuples (or RAMs) per discriminator. The particular mapping is identified by these parameters together with the 'seed' used to initialise the programmed pseudo-random number generator. The mapping is displayed on the graphics terminal as a matrix of the same dimensions as the patterns for which it is intended. Each element of the matrix is the number of the n-tuple to which the corresponding pattern element is connected. The display may be amended to show the pattern elements belonging to just one specified n-tuple, which circumvents some of the problems arising when a one-to-many mapping is in use.

The entire mapping may be sent to an output file as a whole, or individual n-tuples may be selected and written out as desired in order to build up a mapping piecemeal. In this way, n-tuples from several different mappings may be collated into a new mapping. MAPPIT further provides the ability to change a particular n-tuple

by re-defining the pattern elements connected to it. Complete mappings or individual n-tuples may be printed out in legible form, as in Fig.4.7. The left-hand table shows a complete random mapping, with the number of the n-tuple (0 to 23) to which each pattern element is connected. In this case the matrix size is 12x16 elements, and the n-tuple size is 8. The right-hand table shows a single n-tuple No.7, selected from the same complete mapping.

(ii) HAMAND. This program provides the means to analyse sets of patterns on a Hamming distance basis. For each pattern class, it maintains a set of accumulators which stores the frequency of occurrence of each pattern element. This accumulated pattern may be displayed as a matrix in which the brightness of each element is dependent upon its occurrence frequency. Also the occurrence frequencies themselves may be printed out as a matrix. Furthermore, from the accumulated pattern is derived an average pattern in which a particular element is set only if its corresponding occurrence frequency is at least half of the total number of patterns accumulated. Thus for every class an average pattern is defined.

In making Hamming distance analyses, the measurement is normally made between the incoming new pattern and the average patterns of the same class. After analysing a complete set of patterns in this way, HAMAND displays the results in the form of a histogram showing occurrence frequency plotted against Hamming distance. As a further facility, HAMAND allows the selection of patterns whose Hamming distances (from the appropriate average patterns) lie within a pre-selected range.

(iii) FOSOC. This is used for investigating the states seen by the n-tuples of a pattern classifier when presented with a set of patterns. Due to storage limitations it is able to handle only one class of

6	13	22	15	20	20	21	17	23	13	7	18
17	8	11	14	9	4	14	3	8	15	22	18
1	5	22	6	18	3	2	0	0	12	5	5
5	12	18	0	4	14	2	16	5	9	15	15
1	1	2	4	21	14	3	2	22	10	13	11
4	6	23	5	20	7	6	11	10	12	13	7
12	11	20	17	8	8	12	11	15	21	14	13
4	18	0	17	10	17	0	14	19	7	5	2
16	4	19	19	2	23	12	13	7	3	14	4
12	11	21	20	20	3	21	22	9	15	7	22
8	1	16	7	6	10	22	23	3	22	19	17
5	14	1	23	13	23	19	6	18	12	17	0
3	23	7	18	9	0	6	21	16	16	1	6
13	3	8	1	15	11	19	11	1	18	2	23
20	10	0	9	9	2	21	16	4	8	8	10
19	15	17	16	21	20	9	16	10	9	10	19

Complete random map.

.	.	.	.	.	.	.	.	.	.	7	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	7	.	.	.	.	.	7
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	7	.	.
.	.	.	.	.	.	.	.	7	.	.	.
.	.	.	.	.	.	.	.	.	.	7	.
.	.	.	7	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	7	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.

N-tuple no. 7 selected.

Fig. 4.7 Map Display.

pattern at a time, so the normal usage procedure is as follows. Firstly the usual parameters are set up, i.e. pattern dimensions, n-tuple size, and number of n-tuples per discriminator. Then the storage is cleared of any previous contents, and the appropriate mapping read in from the map file. The test patterns are then read in, and for each one the usual mapping process is carried out (see section 3.3.4). However, the storage location addressed within each n-tuple by its own state is not one bit as in the normal classifier, but is a complete word of 16 bits. The teach process involves incrementing this word rather than setting it, and thus a complete record of the frequency of occurrence of every state of every n-tuple is built up. When all the patterns of one class have been processed, the 'discriminator' contents are dumped on an output file (with the extension 'FRE'), then cleared ready to accept the next class. When all classes have been processed, the file is closed, and control passed to a further program, FRAN, for analysis of the results.

(iv) FRAN. As stated above, this program reads the output file, produced by FOSOC, which contains the n-tuple state occurrence frequencies for every discriminator over a complete set of patterns. In its full analysis option, FRAN processes each n-tuple in turn, printing out the occurrence frequency of every state seen by each discriminator (omitting unseen states), and also a list of those states of the n-tuple which were seen by more than one discriminator. At the same time, records are kept of the total number of states seen, the number of different states, the number of states common to more than one discriminator, and the number of patterns causing the n-tuple to go into one of these multiple states (this number is referred to as the number of rejections, since, on its own the n-tuple offers no discriminatory powers for patterns giving rise to one of

these states).

After all the n-tuples have been processed in this way, FRAN prints a table of the running totals listed above. In its concise analysis option, FRAN prints out only this table.

Chapter 5

Character Recognition Experiments

5.1 Introduction

Previous chapters have listed the necessary attributes of a character recognizer for use in a blind reading aid having a hand-held input probe and a letter-by-letter audible output. It is the purpose of the present chapter to show that the n-tuple character recognizer, described previously, is capable of fulfilling these requirements.

The experiments are presented in three parts: firstly those concerned with inherent variations in the source characters, due to case (upper- or lower-) and font. There is a preliminary calibration section in which the various styles of character are tested separately to provide a reference point for the later experiments combining upper- and lower-case characters, and the multi-font experiments. Secondly there is a series of experiments on pattern variations arising from the hand-held input device, namely variations in position due to the manual tracking, and adjacency effects arising because the characters are not seen in isolation. The final set of experiments is devoted to attempts to improve the performance of the recognizer using fixed sets of patterns and mainly being constrained to a fixed amount of storage. These latter experiments are applicable to a wider field than the blind reading aid, since they develop techniques suitable for any n-tuple recognizer of the type discussed here. The term 'optimisation' is used loosely in the text to refer to any means of increasing the recognition ability: it does not imply that the achievement of optimum performance is guaranteed.

5.2 Upper-case, Lower-case and Multifont Recognition

5.2.1 Preliminary Calibration

The type styles chosen for the multifont recognition experiments were:

(a) Folio Bold. A heavy, modern sans-serif font.	**ABC abc**
(b) Times New Roman. A very common old-style serif font. . . .	ABC abc
(c) Clarendon Medium. A common new-style serif font.	**ABC abc**
(d) Typescript. A common type-face, using a half-worn ribbon.	ABC abc

For the remainder of this chapter type styles will be referred to by the appropriate identification, a) to d) as above, with 'UC' for upper-case and'LC' for lower-case.

For calibration purposes, each of the above type-faces was tested in both UC and LC alphabets, using a range of training set sizes, but with a fixed n-tuple size of 8, fixed window dimensions of 12x16 bits, and the same random mapping. The results are summarized in Table 5.1. The corresponding learning curves are plotted for font (c) only, UC and LC, in Fig.5.1. It can be seen that the UC alphabet is easily accommodated by the recognition system in use, but the LC alphabet shows the onset of saturation. This is discernable to a lesser extent for fonts (a) and (b) in Table 5.1. On inspection of the patterns in the LC training and testing sets it was found that the smaller letters were occupying only about half of the available window space, the remainder staying blank most of the time. This is wasteful of the recognizer resources, since all of the n-tuple elements connected to the blank areas are of little use in discriminating between patterns. Furthermore it was found that in some cases the camera resolution was insufficient, causing several groups of letters to be indistinguishable, for example e and c, and l, t and f.

The solution to both of these problems is obviously to increase the camera resolution, so that the LC letters fill the window more fully,

Table 5.1 Recognition Performance for Fonts (a),(b),(c),(d).

Font		% Correct - trained on 200/class.	Trained for max. correct		Trained on all patterns	
			% Correct	Patts/class	% Correct	Patts/class
(a)	UC	96.4	98.8	360	98.8	400
	LC	85.5	87.9	340	84.2	800
(b)	UC	92.9	94.2	360	94.2	400
	LC	78.8	83.2	320	82.6	400
(c)	UC	95.5	98.0	400	98.8	400
	LC	83.0	83.0	200	80.3	400
(d)	UC	94.5	96.3	400	96.3	400
	LC	86.6	97.3	560	97.0	800

All 12x16 bits, n=8, R1216 map 0, test sets all 200 patterns/class.

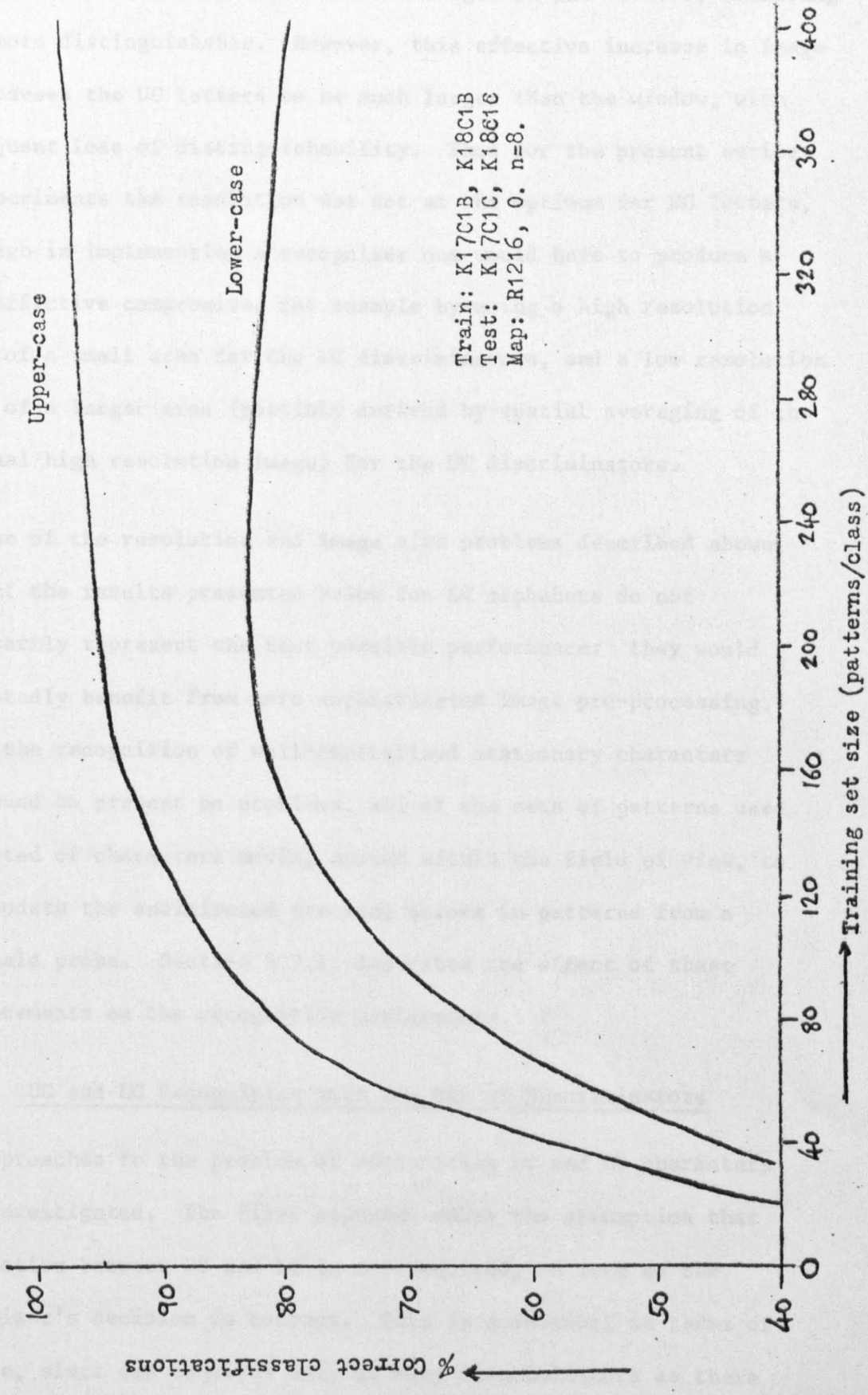


Fig. 5.1 Font (c) Upper-case and Lower-case.

at the same time giving more detailed images of the letters, rendering them more distinguishable. However, this effective increase in image size causes the UC letters to be much larger than the window, with consequent loss of distinguishability. Thus for the present series of experiments the resolution was set at the optimum for UC letters, although in implementing a recognizer one would have to produce a more effective compromise, for example by using a high resolution image of a small area for the LC discriminators, and a low resolution image of a larger area (possibly derived by spatial averaging of the original high resolution image) for the UC discriminators.

Because of the resolution and image size problems described above, many of the results presented below for LC alphabets do not necessarily represent the best possible performance: they would undoubtedly benefit from more sophisticated image pre-processing. Since the recognition of well-centralized stationary characters was found to present no problems, all of the sets of patterns used consisted of characters moving around within the field of view, to accommodate the anticipated tracking errors in patterns from a hand held probe. Section 5.3.1. describes the effect of these displacements on the recognition performance.

5.2.2 UC and LC Recognition with One Set of Discriminators

Two approaches to the problem of recognizing UC and LC characters were investigated. The first approach makes the assumption that distinction between UC and LC is not required, as long as the recognizer's decision is correct. This is economical in terms of storage, since one requires only as many discriminators as there are decision classes, i.e. 26 for the letters of the alphabet. The drawback is that each discriminator must be trained on both UC and LC letters which in most cases bear little resemblance to each other, with the result that saturation of the storage is more likely to occur.

To test the feasibility of this approach, an experiment was conducted in four parts: one for each of the type faces (a) to (d). In each case the recognizer was first trained on UC letters and tested on both UC and LC letters. The store was then reset and the recognizer re-trained on LC letters, and tested on the same sets of UC and LC letters. Finally, the recognizer was trained on both UC and LC letters, and again tested on the same sets. The results for all four fonts are presented in Table 5.2. The figures in parentheses represent the change in performance due the extension of the training sets.

The results are encouraging, since they show that the discriminators are capable of accommodating both UC and LC fonts with little degradation of performance. An interesting feature in Table 5.2 is that the recognition performance on LC patterns when the system has been trained only on UC patterns, and vice versa, is considerably better than one would expect for purely random decisions (a figure of about 4% correct classifications). This indicates the presence of some similarities between UC and LC characters of the same class (e.g. 'O' and 'o', 'C' and 'c', etc.)

The training sets used in this experiment were all 200 patterns per class, and it is likely that the performance figures could be improved slightly by paying closer attention to the choice of training set sizes. However, the point of the experiment has been made: that it is feasible to use the same set of discriminators for both UC and LC characters.

5.2.3 UC and LC Recognition with Two Sets of Discriminators

The alternative approach to UC and LC recognition is to use duplicate sets of discriminators: one set trained on UC letters, and the other on LC letters. This avoids any saturation problems which may arise

Table 5.2 UC and LC Recognition with One Set of Discriminators.

Font	Trained on UC		Trained on LC		Trained on UC and LC	
	% UC Correct	% LC Correct	% UC Correct	% LC Correct	% UC Correct	% LC Correct
(a)	96.1	15.6	15.8	84.5	93.5 (-2.6)	82.5 (-2.0)
(b)	92.1	18.2	14.8	82.7	89.5 (-2.6)	77.6 (-5.1)
(c)	97.4	16.6	14.2	82.7	94.2 (-3.2)	76.8 (-5.9)
(d)	91.8	16.0	17.3	91.2	89.2 (-2.6)	90.2 (-1.0)

All 12x16 bits, n=8, R1216 map 0. Test sets: 50 patts/class. Training sets: 200 patts/class.

using the technique suggested in the preceding section, since the two sets of probability density functions are now completely independent. The drawback, however, is that the storage requirements are doubled, and for this reason the experiment performed using this approach, for fonts (a) to (c), was limited to the 26 classes A - M (UC) and a - m (LC). Normally 52 discriminators would be necessary.

The experiment was performed in a similar manner to that of the preceding section, i.e. for each font the recognizer was first trained on UC letters and tested on both UC and LC letters; then it was trained on LC letters only, and tested in the same way; finally the recognizer was trained on both UC and LC letters, and again tested on the same sets of UC and LC letters. The results are presented in Table 5.3, the figures in parentheses representing the changes in performance due the extension of the recognizer.

As expected, this approach yields acceptable results, since both UC and LC characters are recognized with little degradation of performance when both sets of discriminators are operating simultaneously. Overall, the results are slightly better than for the first approach, using only a single set of discriminators, though it must be borne in mind that this has been achieved by doubling the storage requirements, and also that only half of the total number of classes has been used: extension to 52 discriminators would certainly cause some degradation of performance. Again it should be noted that the training sets used are not necessarily of optimal size.

5.2.4 Multifont Recognition

The problem of recognizing a range of different type faces is similar in some respects to that of recognizing UC and LC versions of one font, in that each class of character can exist in one of several distinct forms. The approach to the problem thus develops along

Table 5.3 UC and LC Recognition with Two Sets of Discriminators.

Font	Trained on UC		Trained on LC		Trained on UC and LC	
	% UC Correct	% LC Correct	% UC Correct	% LC Correct	% UC Correct	% LC Correct
(a)	98.9	14.3	14.9	86.0	93.4 (-5.5)	85.8 (-0.2)
(b)	93.2	14.9	17.8	83.5	89.7 (-3.5)	82.0 (-1.5)
(c)	98.3	14.9	13.2	86.6	96.3 (-2.0)	85.1 (-1.5)

All 12x16, n=8, R1216 map 0. Test sets: 50 patts/class. Training sets: 200 patts/class.
UC = classes A-M; LC = classes a-m.

similar lines, i.e. one may train the discriminators on a number of different type faces and accept a slight degradation of performance, or one may reserve separate sets of discriminators for each font, and accept the multiplication of the storage requirements. Since the experiments reported in section 5.2.2 show that a single set of discriminators may be usefully trained on both UC and LC characters, and since the amount of pattern variation between different fonts is much less than the difference between UC and LC versions of the same character, it was decided to adopt the first approach.. In other words, each discriminator was trained on patterns belonging to several different fonts.

The first experiment was on UC characters only from fonts (a), (b), (c). There were seven parts to the experiment, after each of which the recognizer was tested on sets of characters from each of the fonts. The first three parts involved training the recognizer separately on each of the three type faces. The next three parts involved training on three different pairs of type faces, and the last part involved training on all three type faces. The results are presented in Table 5.4. Each figure in parentheses represents the change in performance for a particular type face when the training set has been extended to include one or more extra type faces. The second experiment, the results of which are presented in Table 5.5, was a repetition of the first, using LC characters instead of UC.

The results for UC characters are good, since they show that the recognizer is capable of accommodating three different type faces simultaneously with little degradation of performance (in fact the performance on font (b) shows an improvement - possibly indicating that the original training set is not adequate). The degree of similarity between the fonts can be determined by inspecting the recognition performance on a font not included in the training set. The results

Table 5.4 Multifont Recognition, UC.

Training set(s)	% font (a) correct	% font (b) correct	% font (c) correct
Font (a)	96.1	54.4	67.8
Font (b)	57.3	92.1	66.8
Font (c)	74.9	63.1	97.4
Fonts (a),(b)	95.1 (-1.0)	94.6 (+2.5)	78.7
Fonts (a),(c)	95.6 (-0.5)	64.2	96.5 (-0.9)
Fonts (b),(c)	73.6	92.3 (+0.2)	97.7 (+0.3)
Fonts (a)(b)(c)	93.3 (-2.8)	93.2 (+1.1)	96.4 (-1.0)

All 12x16, n=8, map 0. Test: 50/class. Train: 200/class.

Table 5.5 Multifont Recognition, LC.

Training set(s)	% font (a) correct	% font (b) correct	% font (c) correct
Font (a)	84.5	33.9	51.3
Font (b)	30.8	82.7	53.3
Font (c)	48.8	50.3	82.7
Fonts (a),(b)	73.0 (-11.5)	74.4 (-8.3)	60.1
Fonts (a),(c)	75.2 (-9.3)	51.8	78.0 (-4.7)
Fonts (b),(c)	45.2	73.5 (-9.2)	74.9 (-7.8)
Fonts (a)(b)(c)	64.5 (-20.0)	65.9 (-16.8)	68.2 (-14.5)

All 12x16, n=8, map 0. Test: 50/class. Train: 200/class.

for LC characters show a marked fall in performance when the training sets are extended. This is due to the problem encountered in section 5.2.1, namely that the camera resolution is insufficient for adequate identification of some of the smaller characters. It is expected that increasing the resolution would give results comparable to those in Table 5.4, since LC letters are not inherently any less recognizable than UC letters.

This experiment shows that fairly similar type faces can be accommodated merely by extending the training sets, and without changing the structure of the recognizer. Extension to a wider range of fonts might require the provision of a further set of discriminators, particularly where the fonts are very dissimilar in size or style. In such a system, recognition could be assisted if one had prior knowledge of the type face in use - one could simply switch in the appropriate set of discriminators.

5.3 Tracking and Segmentation

5.3.1 Tracking

In Chapter 3 it was shown that the character recognizer for a blind reading aid must be capable of accommodating the pattern variations arising in the camera and digitizing systems, namely random electrical noise, distortions due to the digitizing matrix, and variation in the position of the characters within the field of view. The latter is especially important since a hand-held probe cannot be expected to track along a line of text with any great degree of accuracy.

The extent of horizontal misalignment of characters is a function of the camera frame rate and the speed at which the user is scanning the line of text. Consequently by using a high enough frame rate one may always ensure that all characters will be seen within a certain distance

from the centre of the field of view. For example, suppose that the average width of characters is 1 mm, and the resolution in use is 5 bits per mm, i.e. 5 bits per character, If the text is being scanned at 10 mm per second, i.e. 10 characters per second, the frame rate must be at least 50 frames per second to ensure that each character is seen near the centre of the field of view.

Vertical misalignment, however, is more difficult to accommodate since the vertical motions of the hand-held probe are irregular, unlike the horizontal scanning motion. The simplest approach, and the one adopted in the simulations, is to permit the input matrix of the recognizer - the 'window' - to move vertically under programmed control in order to keep the line of text in view. The way in which this occurs is described in section 4.4.3. Since the complete frame from the camera is 64x64 bits and the window is typically 12x16 bits, a considerable degree of drift tolerance is possible. In a practical implementation one would require some form of audible warning whenever the window approaches an edge of the frame, so that corrective action may be taken by the user. It is interesting to note that the rate of vertical movement of the window is limited only by the cycle time of the recognizer - the time required for a decision to be made about the class of the current input pattern. In a microcircuit read-only memory implementation the cycle time would be of the order of microseconds, so that the tracking action of the window would be completely undiscernable by the user. The rate of horizontal scanning (by hand) is of course limited by the frame rate of the camera.

Inspite of the rough centralization of characters by the horizontal scanning action and the vertical tracking action, it is still necessary for the recognizer to handle patterns in which the characters are displaced by several bits in any direction from the centre of the

window. Table 5.6 summarises the results of an experiment in which the recognition performance was tested for various sets of characters displaced by differing amounts from the centre of the field of view (d is the number of bits of displacement in any direction).

Table 5.6 Recognition Performance at Various Displacements.

Font	% Correct classifications			
	d=0	d≤1	d≤2	d≤4
(b) UC	100.0	99.9	97.9	94.2

12x16 n=8 R1216 map 0.

Clearly well-centred characters are recognized with little difficulty. Displacements of up to 2 bits are handled adequately, but beyond this the performance begins to fall off. At 4 bits displacement the characters are only just within the field of view: any further displacement would cause them to be truncated. In a practical implementation one would need to accommodate displacements of up to about 3 bits; for the experiments reported in later sections, displacements of 4 bits or more have been used, for example in some of the optimisation experiments, where one does not wish to start off with too high an initial performance.

5.3.2 Segmentation

The term segmentation is normally used for a process applied to patterns

derived from running printed text before they are presented to the pattern recognizer proper, with the aim of isolating the individual letters by removing their neighbours. One way of doing this is to detect the blank vertical strips between the letters, though allowance must be made for ligatures (two letters run together) and spurious ink patches. These techniques are basically sequential in nature, since they involve searching for the gaps, and dealing with the exceptions by means of some pre-programmed algorithm. For the present research, however, it was felt worthwhile to investigate how the n-tuple recognizer, with its fast and highly parallel structure would behave when presented with unsegmented text. In effect, the recognizer is required to perform its own segmentation, by recognizing a character when it is visible, and rejecting the spurious patterns arising when two or more characters are visible simultaneously. This is achieved by setting the confidence threshold (see section 3.3.7) suitably high so as to reject the spurious patterns.

Experiments were performed on characters from font (a) LC since pattern degradation due to resolution effects was not severe in this case (see section 5.2.1). However, the matrix dimensions were reduced to 10x14 bits (from 12x16) in order to minimize the amount of blank space in the patterns (for isolated characters) or the amount of encroachment of adjacent characters into the patterns (in the case of running text). An n-tuple size of 8 was used, with a 1:2 random mapping, giving 34 n-tuples in all. The training set contained isolated examples of each character displaced by up to 2 bits from the centre of the field of view. A calibration test set was used, containing patterns similar to those in the training set, followed by two segmentation test sets, labelled 'SEG1' and 'SEG2' and derived from concatenated strings of characters. Each consisted of two complete scans across the alphabet, with the vertical tracking algorithm

activated so that the characters were positioned in a realistic manner. In SEG1 the characters were separated by a gap of approximately half the average character width, whilst in SEG2 they were separated by less than one quarter of the average character width. Patterns in SEG1 and SEG2 were labelled 'class zero' if there were no characters visible whose centres were displaced horizontally by three bits or less from the centre of the field of view. Two tests were made on each of SEG1, SEG2: one including class zero patterns, the other excluding them.

The results of these tests are presented in Table 5.7, where it can be seen that the choice of threshold is quite important, in order to ensure that class zero patterns are rejected (counting as a correct classification). There is negligible difference in performance between SEG1 and SEG2, but both are significantly worse than the calibration test on isolated characters. On looking at the responses in more detail, it was found that narrow characters such as 'i' and '1' were rejected or mis-classified much more often than the wider characters which filled the field of view more fully. Figure 5.2 shows how the optimum threshold value was chosen for the SEG2 test: a smaller value worsens the performance because more class zero patterns are mis-classified; a larger value does so because more non-class zero patterns are rejected. Note that the performance figure at the optimum threshold value is meaningful only when the proportion of class zero to non-class zero patterns is realistic, i.e. exactly as would be encountered in the intended application. Even so, one should ensure that the threshold chosen does not cause an unacceptable degree of rejection of non-class zero patterns.

Table 5.7 Segmentation Experiments.

Training set	Test set	% Correct classifications		
		Excluding class 0	Including class 0	
			No threshold	Optimum threshold
Isolated characters	SEG1	76.4	26.1	78.2
Isolated characters	SEG2	76.3	25.6	78.6
Isolated characters	Calibration	90.5	--	--

10x14, n=8, 34 n-tuples. Training set: 400 patterns/class. All font (a) LC.

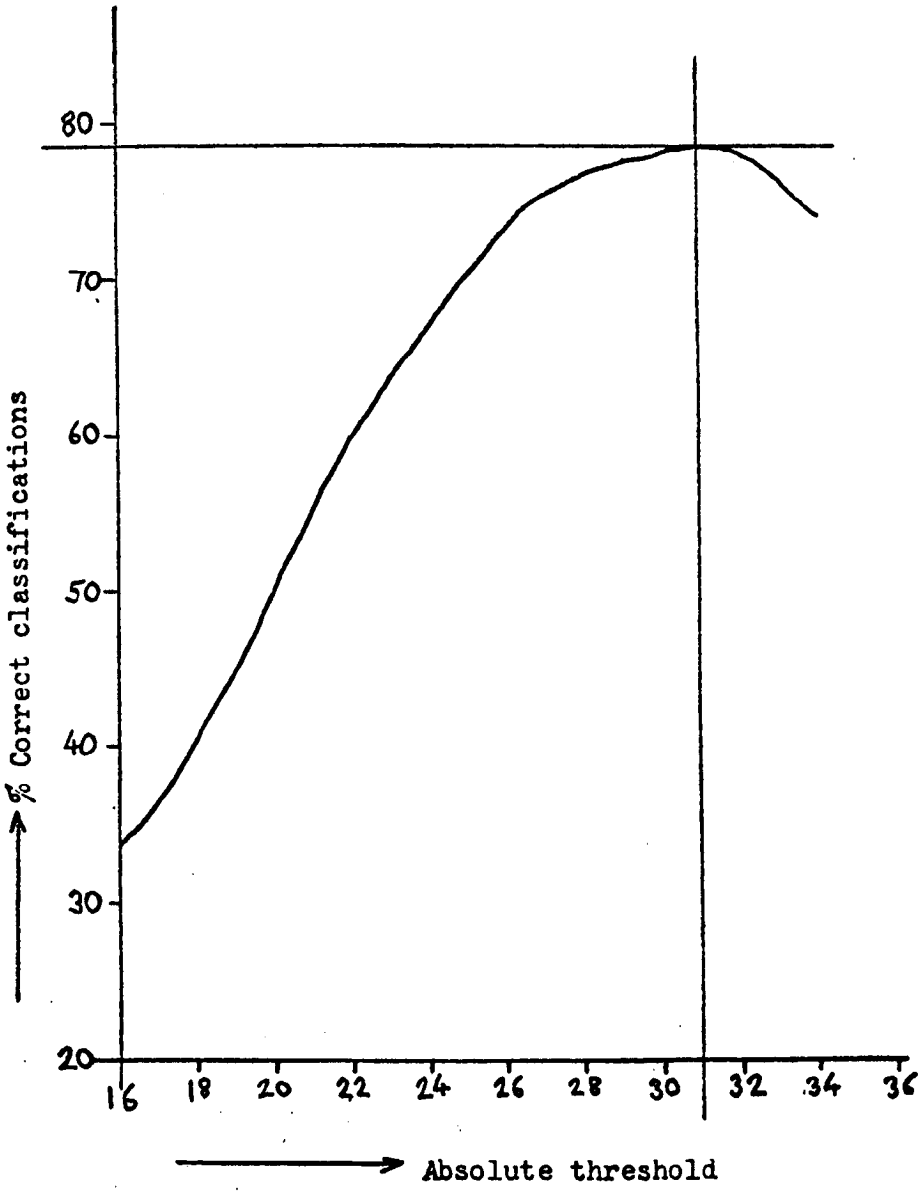


Fig. 5.2 Segmentation Test: SEG2.

The results are encouraging in that they demonstrate the possibility of 'self-segmenting' running text by using the recognizer's confidence in its decisions in order to reject unclassifiable patterns. Some problems do arise with this technique, however. For example when only part of a wide character is visible spurious decisions can arise , especially in the case of 'm' (half of which looks like an 'n') or 'w' (half of which resembles a 'v'). Widening the field of view to avoid such confusions has its own drawbacks because of the intrusion of neighbouring characters in the case of a narrow central character. A suggestion for overcoming this problem is to use specially adjusted mappings which place a greater weight on the centre of the field of view than on the edges. This could be achieved by employing a greater density of n-tuples in the central region.

5.4 Optimisation Experiments

5.4.1 Number of n-tuples

The number of n-tuples employed by the character recognizer is a key factor in its design. In general, and as shown by Ullmann (22), recognition performance improves as the number of n-tuples increases, but with diminishing returns: beyond a certain point there is little to be gained from adding more n-tuples.

Fig.5.3 shows the results of an experiment in which two recognizers, one having 8 n-tuples (n=6), the other having 112 n-tuples, were applied to the same training and testing sets of patterns over a range of training set sizes. It is immediately obvious that the larger system attains a higher level of performance, and that its behaviour is much smoother than that of the smaller system. This is because the smaller system is more sensitive to fluctuations in the quality of the training patterns, since each n-tuple carries much more weight in making the classification decisions than in the larger

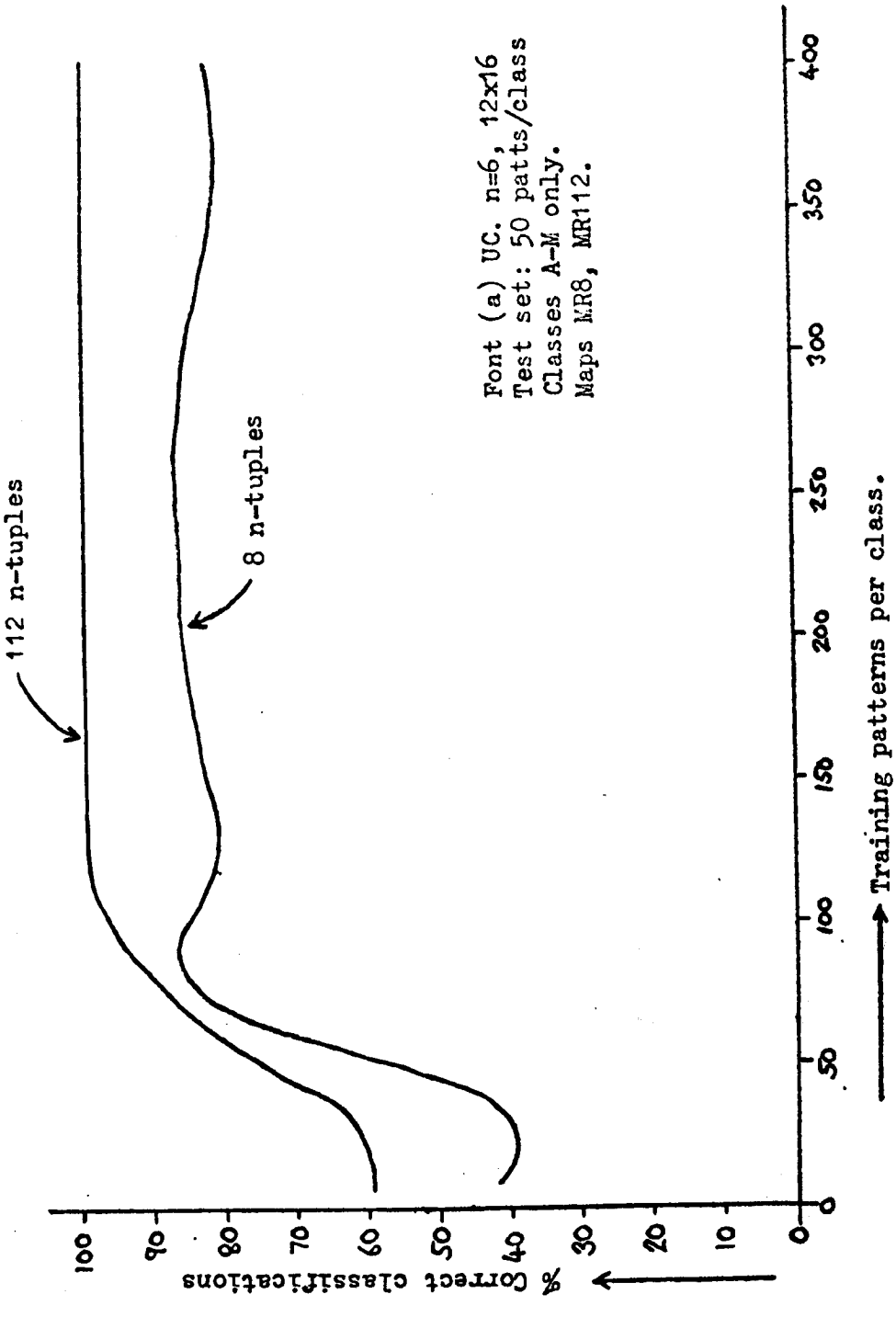


Fig. 5.3 Learning Curves for 8, 112 n-tuples.

system. Thus the existence of a 'rogue' n-tuple state in the training set will cause 1/8 of the n-tuples in the small system to respond erroneously when the state is encountered in a test pattern, whereas only 1/112 of the n-tuples will be affected in the large system. Fig.5.3 further shows that the optimum performance is attained after about the same number of training patterns in both cases. This is to be expected since the rate of filling of the n-tuple stores is the same in both cases: it is independent of the total number of n-tuples.

Further experiments were performed in which the variation of recognition performance with the total number of random n-tuples was investigated for two sets of patterns exhibiting different degrees of within-class variation (and hence difficulty of recognition). The results are presented in Fig.5.4. It is clear that the number of n-tuples needed for the recognition of font (a) UC is about 32, but in the case of font (b) LC the choice is not so obvious, and one must make a trade-off between performance and system size.

It is interesting to compare the curve for font (b) LC in Fig.5.4 (with n = 6) with the calibration results given in Table 5.1 (with n = 8). The 6-tuple system achieves about 87% correct classifications when using 96 n-tuples, i.e., a total of 6144 bits per discriminator. The 8-tuple system achieves about 83% when using 24 n-tuples, i.e. 6144 bits per discriminator. The results however are not directly comparable since Fig.5.4 was derived for only 13 classes (A - M) whereas Table 5.1 used all 26 classes (A - Z). Nevertheless it is clear that one should carefully consider the number of n-tuples to be used when designing a recognizer for a particular application.

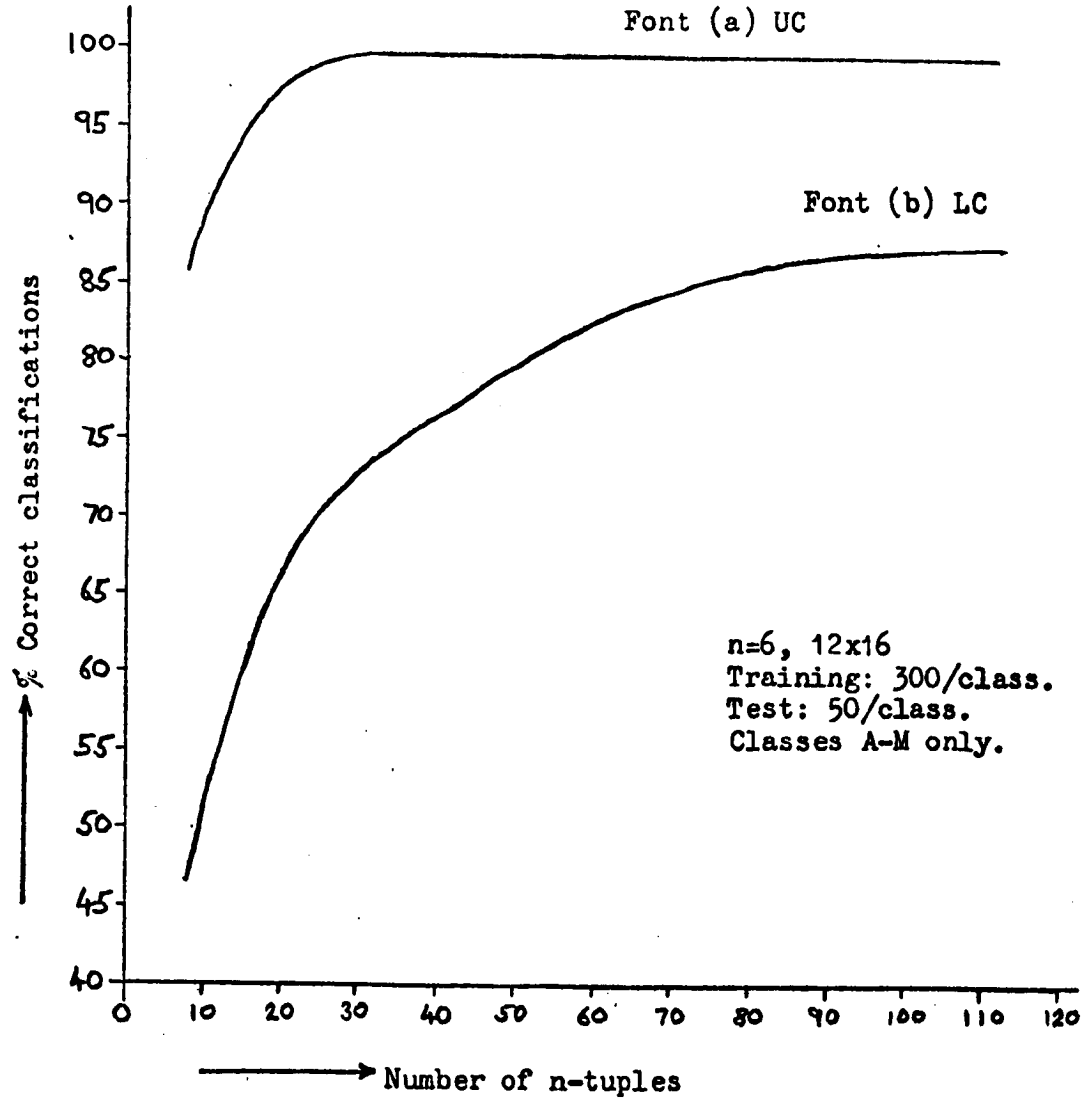
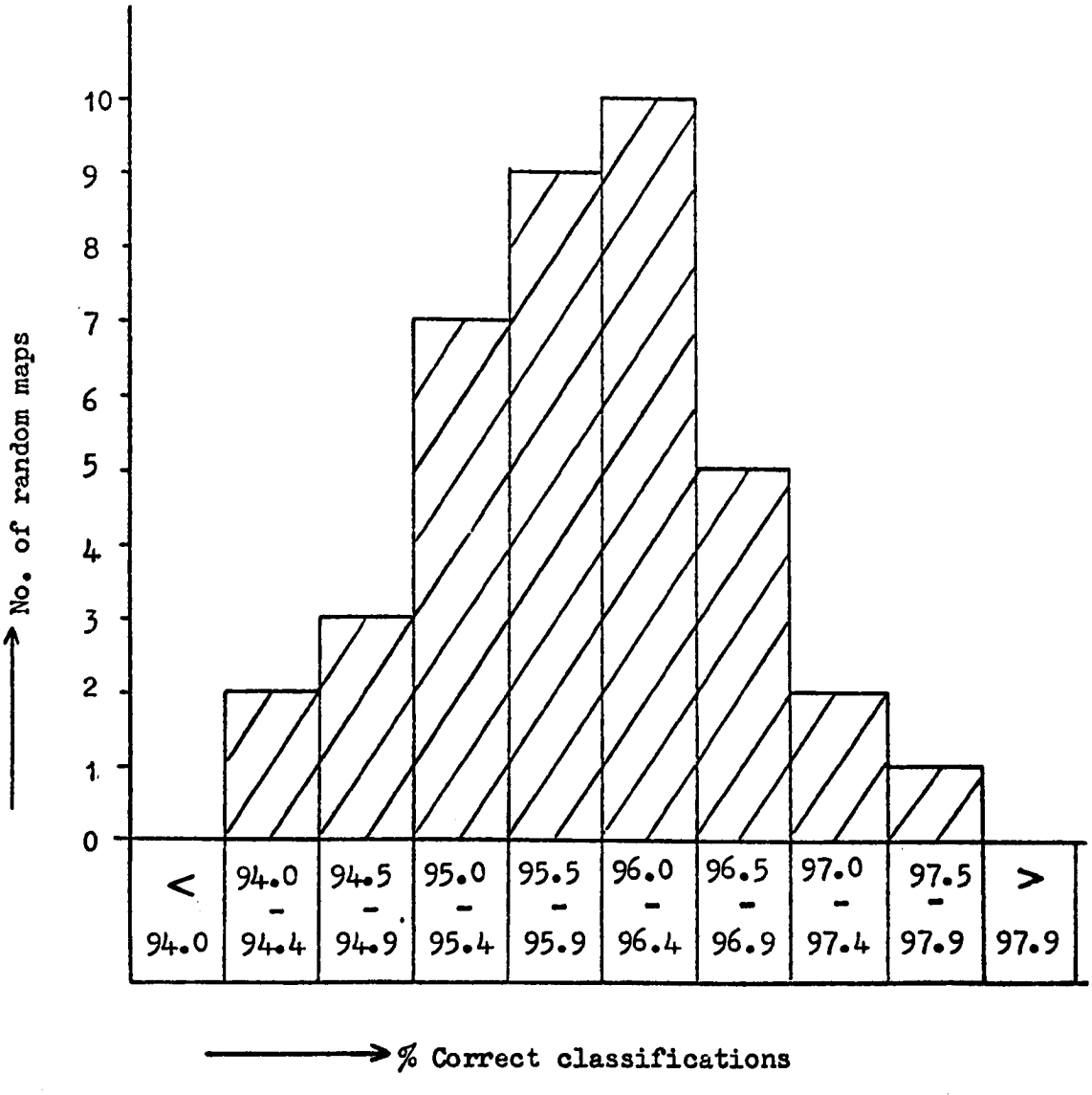


Fig. 5.4 Performance Against Number of n-tuples.

5.4.2 Random Maps

All of the experiments described so far have made use of a single random map (more precisely pseudo-random, but the distinction is not important here), and have not been concerned with any influence the choice of map may have on the performance of the recognizer. **The mapping** is significant in that it determines the way in which the recognizer must discriminate between the pattern classes, i.e., it determines the features (n-tuples) upon which the recognition is to be based. Obviously some n-tuples will be of more value than others, but unfortunately it has proved difficult to find a means of measuring the performance of individual n-tuples, mainly because the recognition ability of the whole system is dependent upon groups of n-tuples acting together. In fact, little improvement has ever been made upon the performance of random maps.

Since different combinations of n-tuples implement their decision surfaces in different ways, one would expect a number of random mappings to exhibit a range of performance values. This is shown in Fig.5.5 for 39 different maps, all using the same sets of patterns. It is clear that the distribution is sharply peaked, as found by Prorok (34) in a similar experiment for a completely different set of patterns. Thus the likelihood of finding by chance a mapping whose performance is very much better than average is extremely small. Furthermore an exhaustive search of all possible mappings is completely impractical: for the configuration used in the above experiment one would need to test over 10^{200} different mappings. The way in which random mappings have been implemented for the experiments reported in this chapter is described in more detail in sections 3.3.6 and 4.4.5.



Font (d) UC, 12x16, n=8. Train: 400/class. Test: 100/class.

Fig. 5.5 Performance Distribution of Random Maps.

5.4.3 Combinations of Random Maps

In the previous section it was shown that the performance distribution of random mappings is very sharply peaked, i.e., there is little point in searching through large numbers of mappings in the hope of finding one giving outstanding recognition performance. However, these performance figures are derived from the total number of test patterns correctly classified and thus give no information about the identities of the patterns being mis-classified. Closer inspection of the responses to the test set reveals that in general different mappings cause mis-classifications of different characters. For example, the features sampled by one particular mapping may be well suited to discriminating between O and Q, for instance, but may be very poor at distinguishing P from F. A second mapping, however, could conceivably operate in exactly the opposite manner, with good P/F discrimination but poor O/Q discrimination. Thus one would expect a combination of the two maps (the first used for the O and Q discriminators, and the second for the P and F discriminators) to show an improvement in performance over the two maps individually.

Fig.5.6 a - f show confusion graphs for six different random maps used individually on training and optimising sets from font (d) L.C. In this case the optimising set was used as a test set in order to establish the identities of the mis-classifications. In the graphs each node represents a character class. A line joining two nodes signifies that characters of the first class were mis-recognized as belonging to the second class. The number of patterns mis-classified is indicated near each arrow. Thus for map 17 (Fig.5.6a) the most important mis-classification is O/Q: 14 characters of class O were classified as class Q, whilst 7 Qs were classified as Os. Map 3 on the other hand exhibits much better performance, with only 4 O/Q confusions and 1 Q/O confusion. Thus one begins to see the advantages

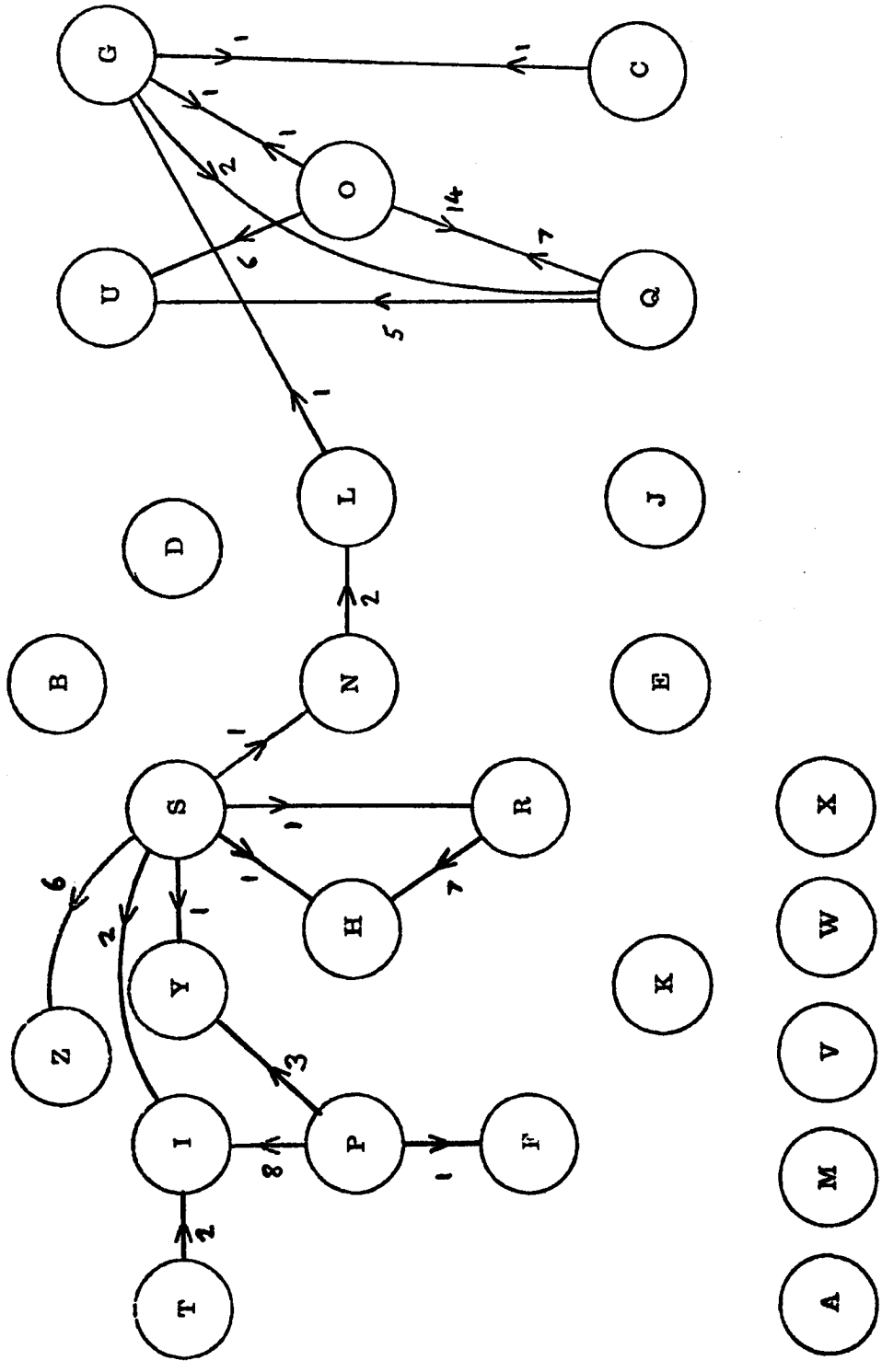


Fig. 5.6a Confusion Graph for Map 17.

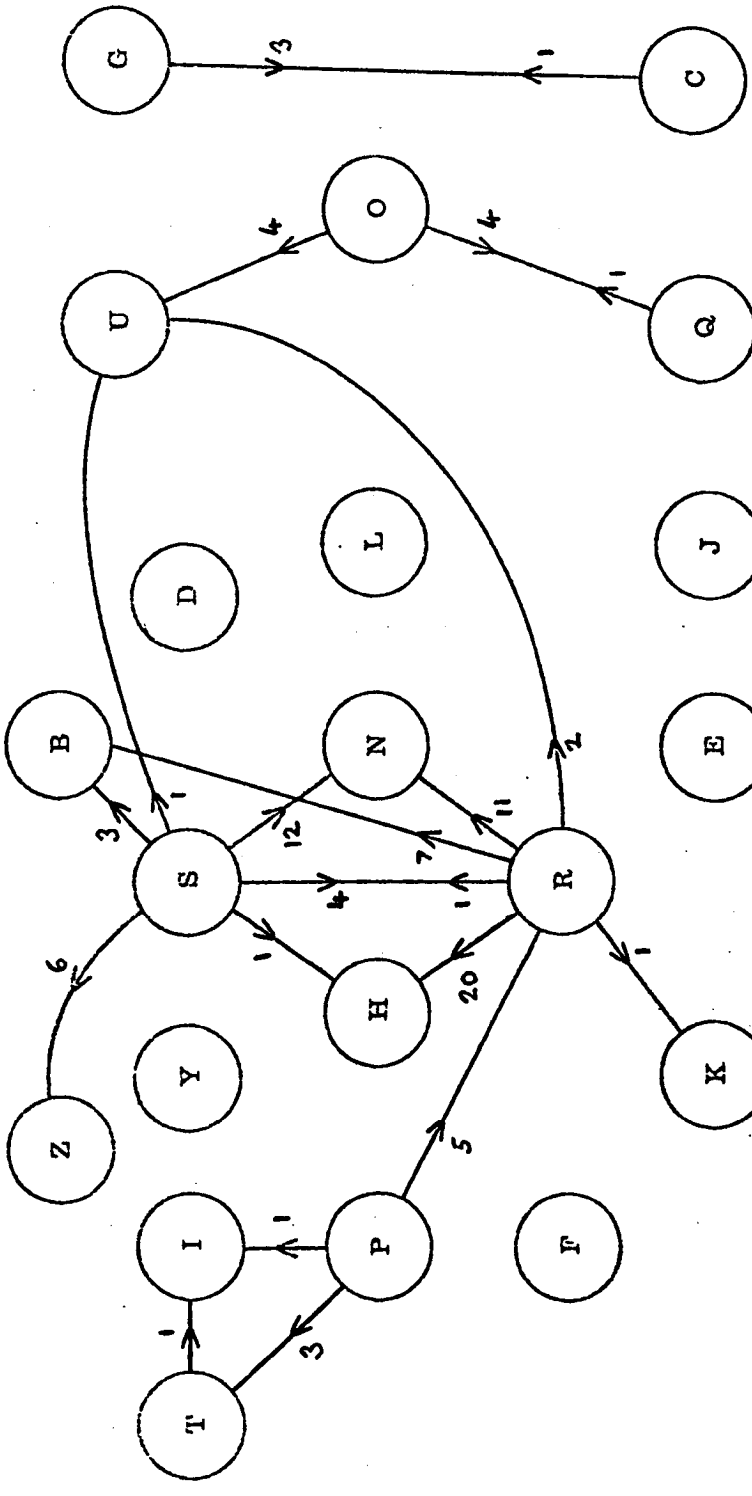


Fig. 5.6b Confusion Graph for Map 3 .

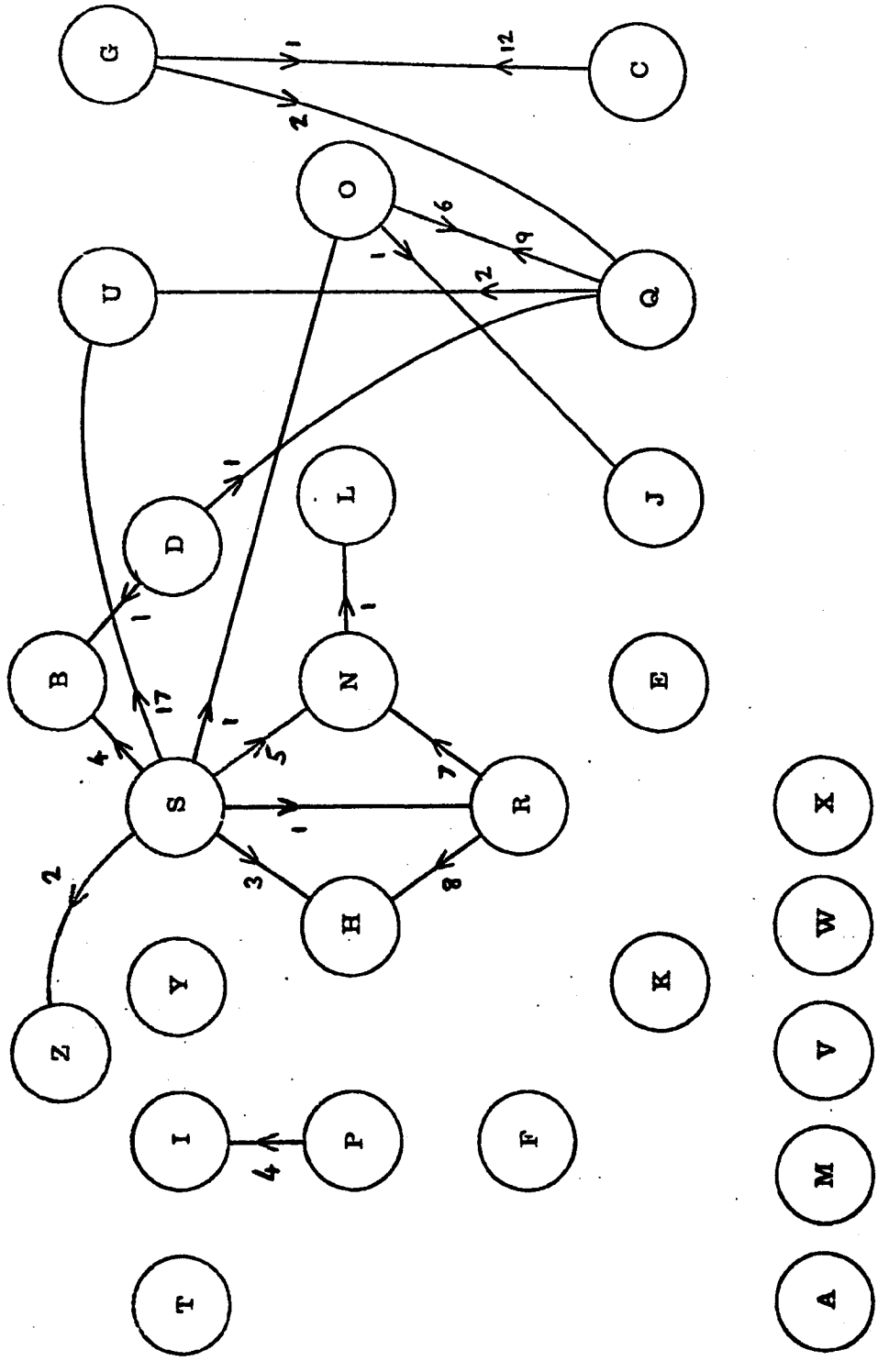


Fig. 5.8c Confusion Graph for Map 24.

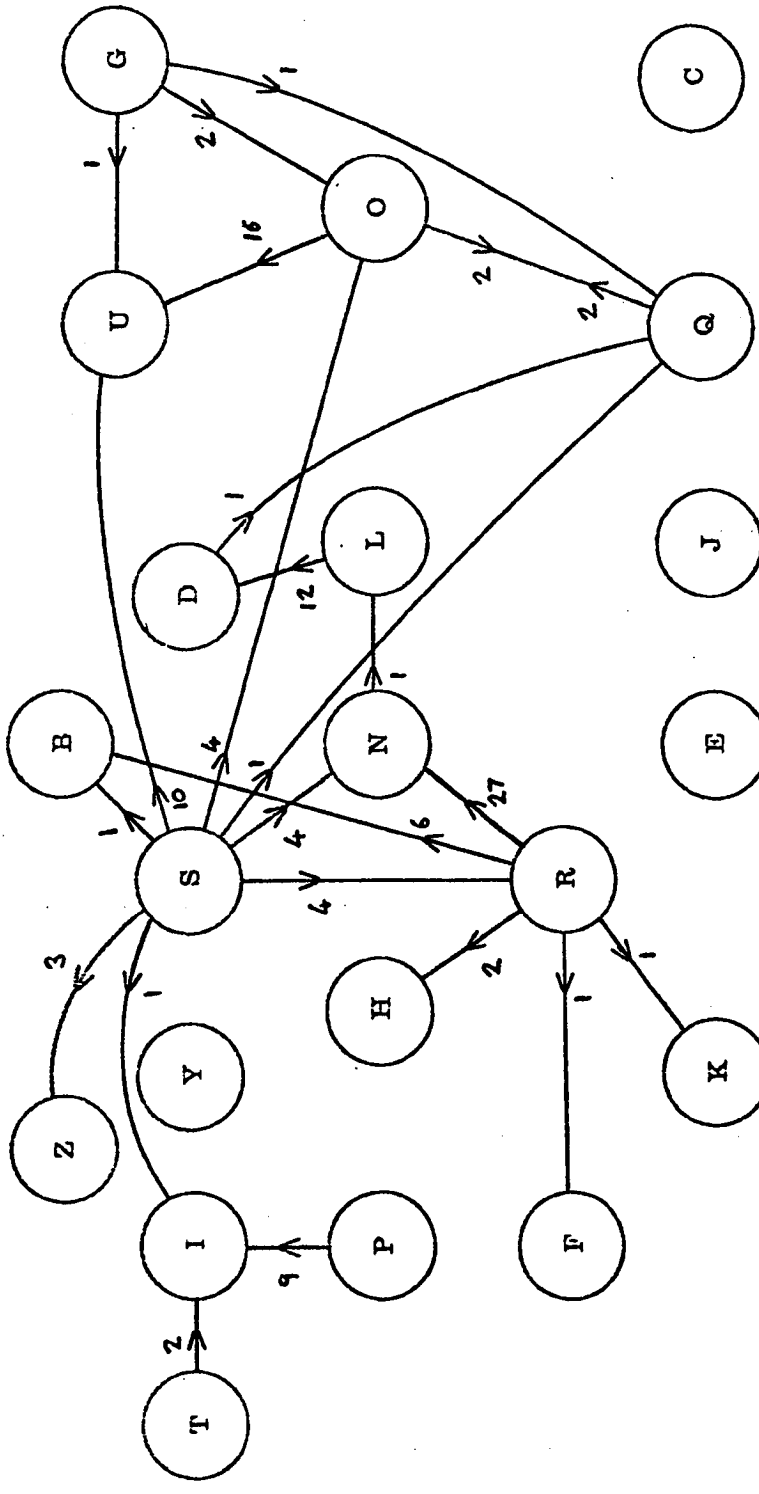


Fig. 5.6 d Confusion Graph for Map 5 .

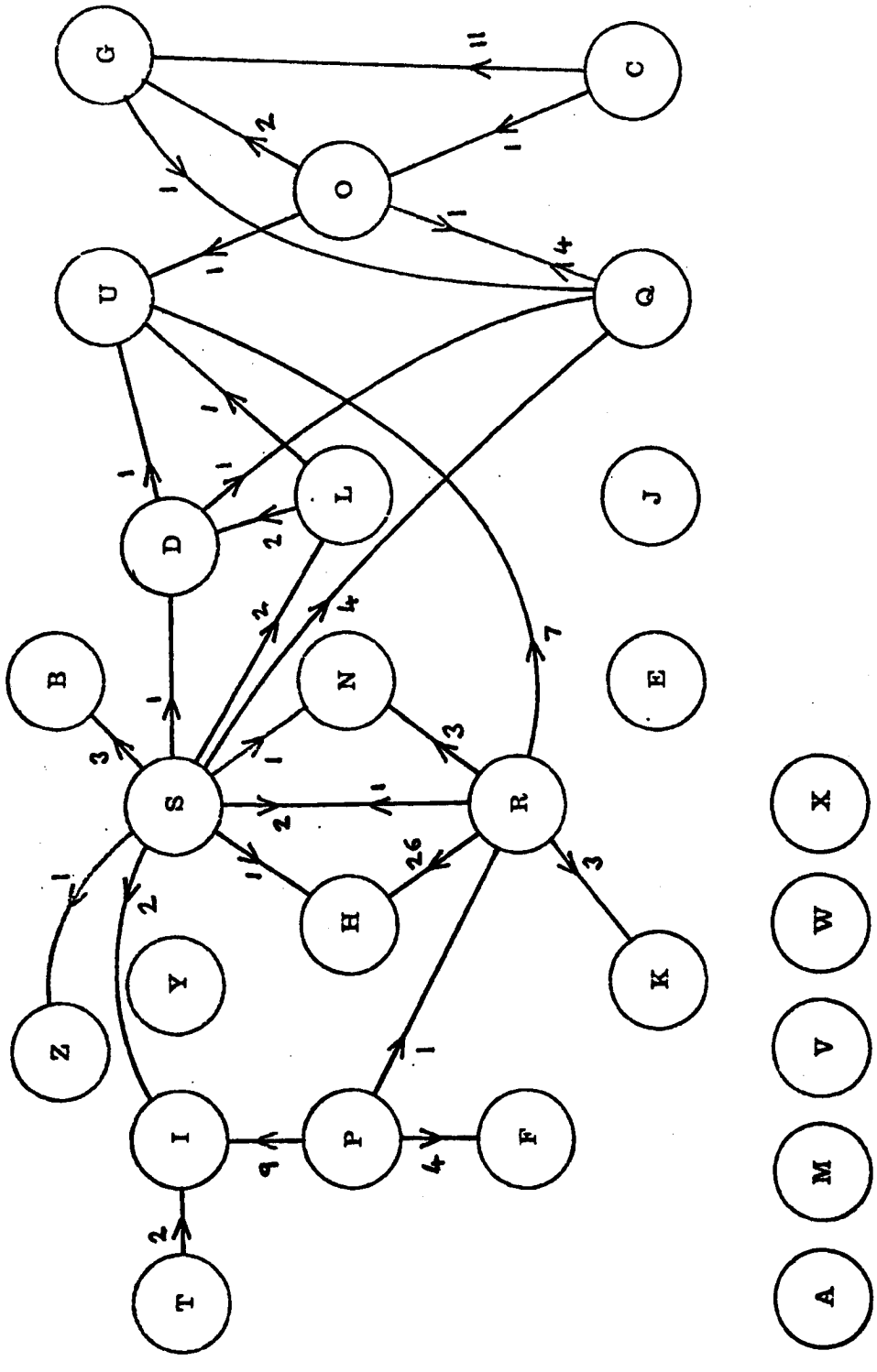


Fig. 5.6 e Confusion Graph for Map 18 .

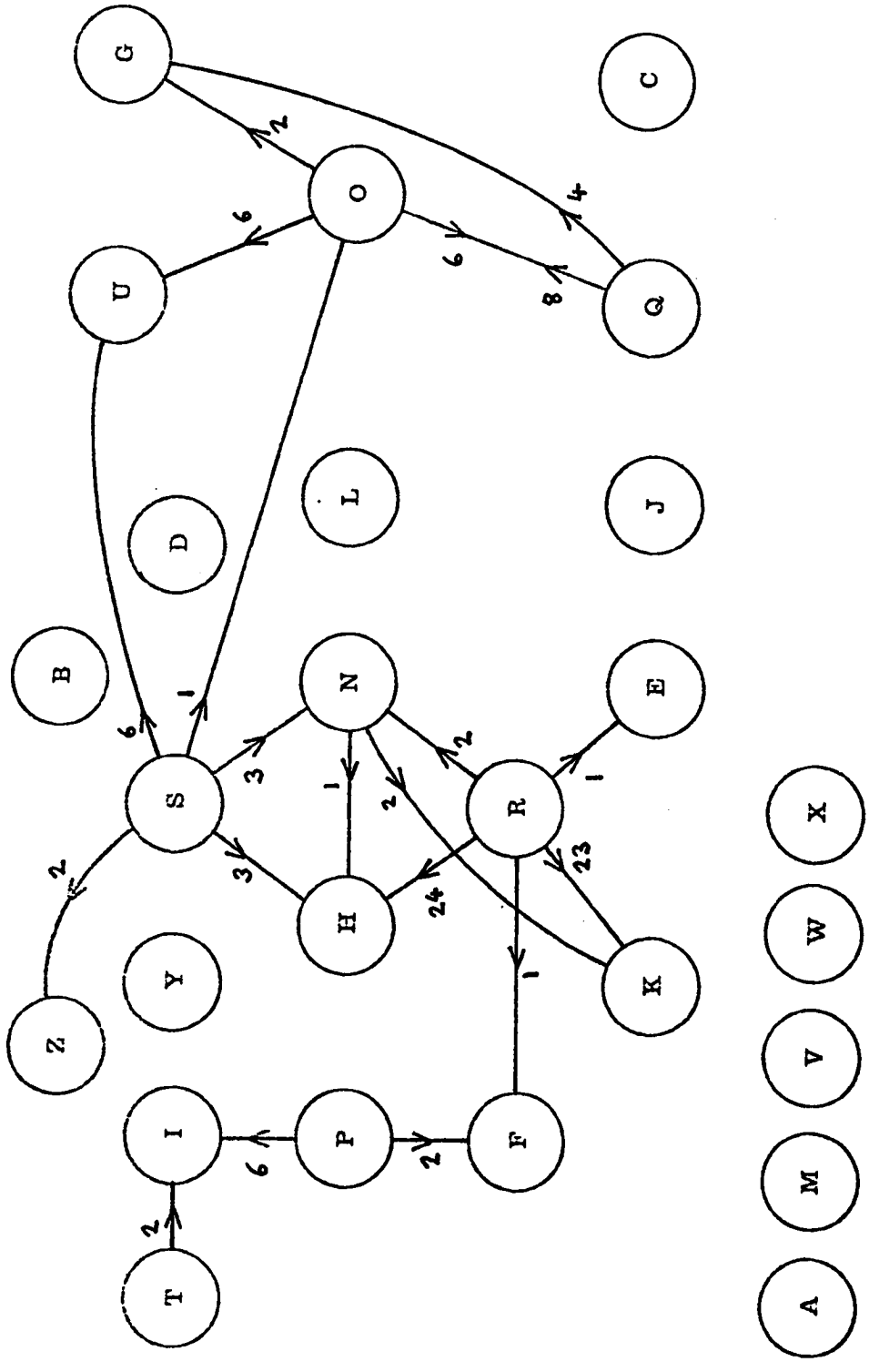
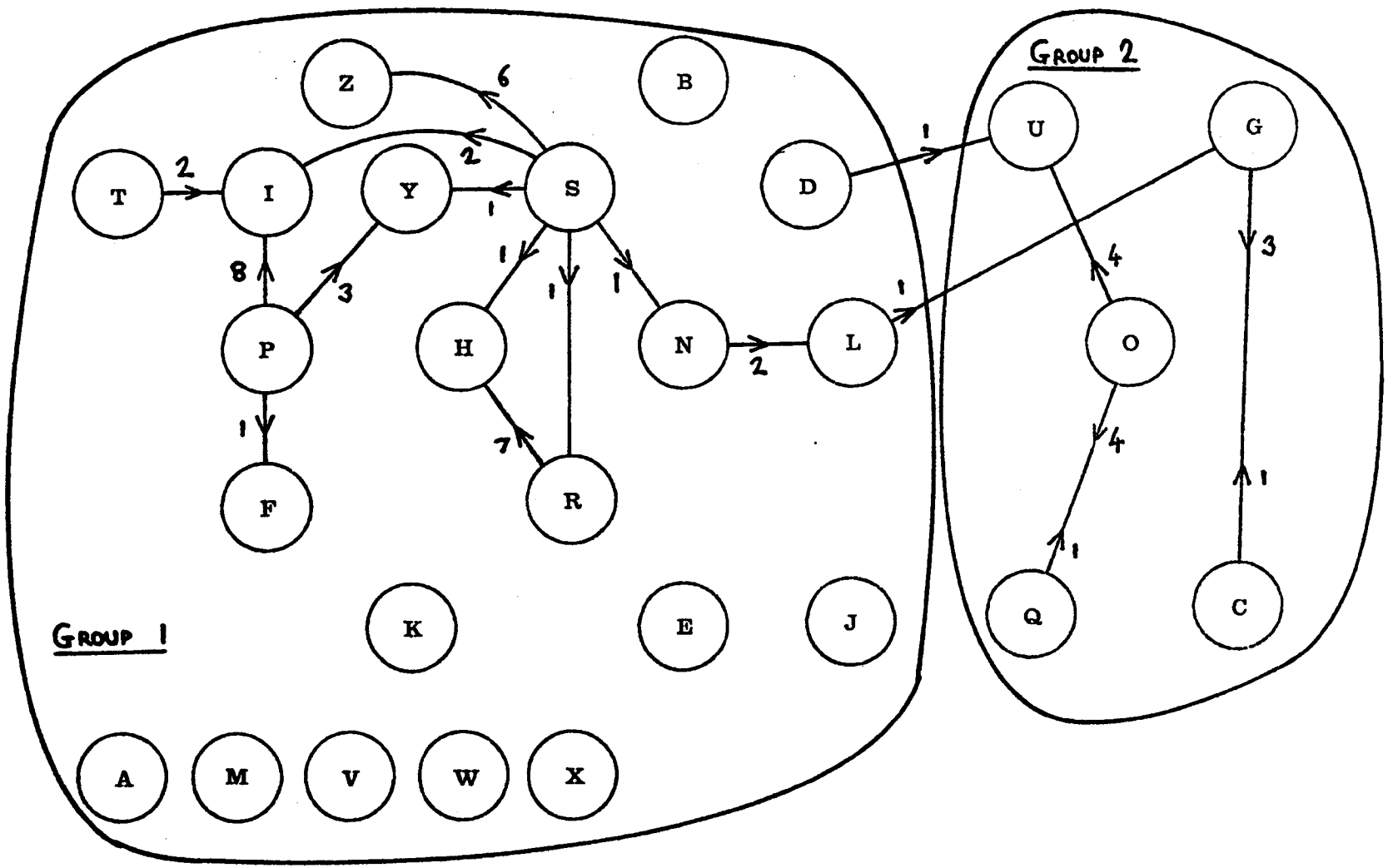
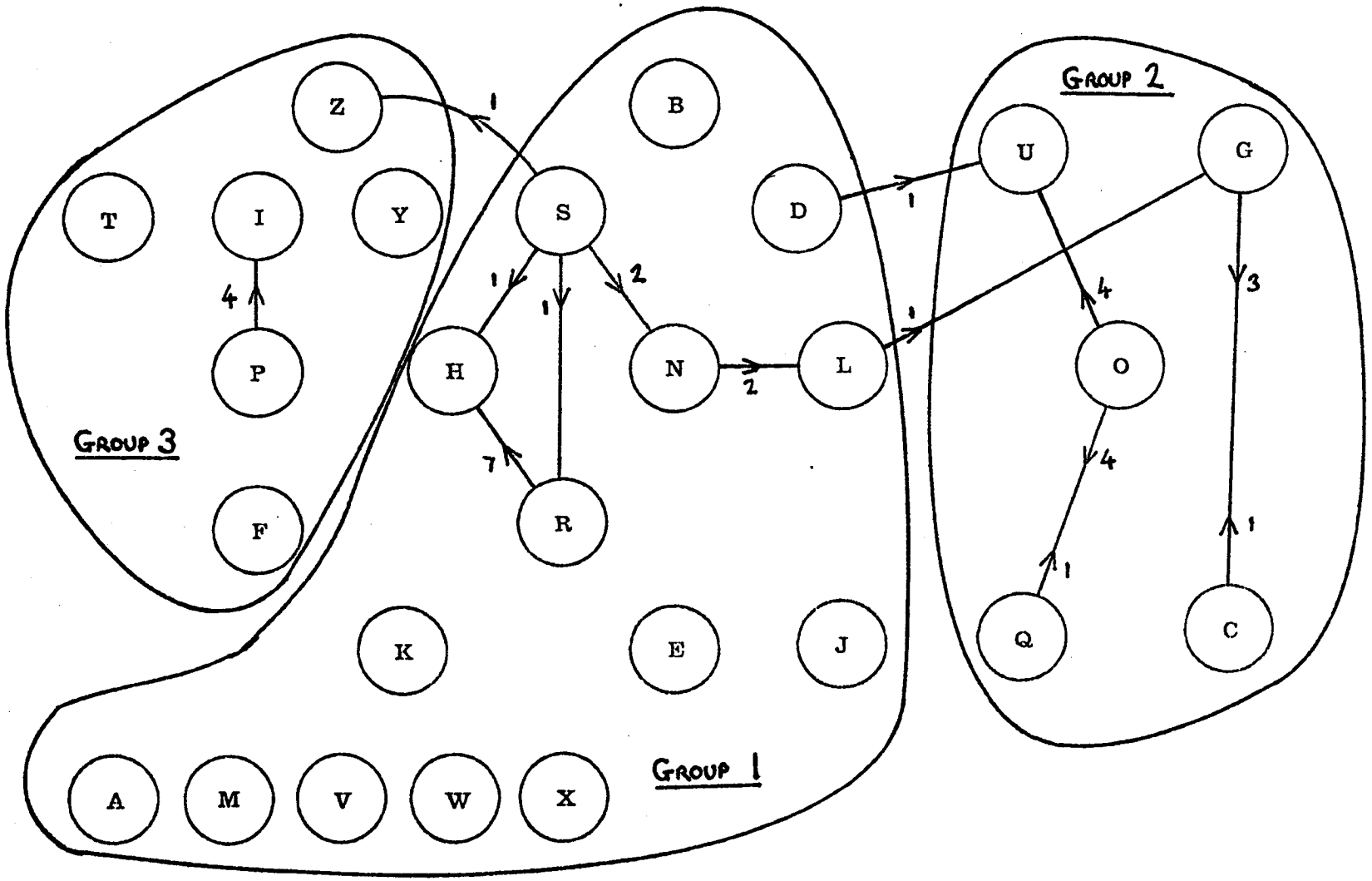


Fig. 5.61 Confusion Graph for Map 27.



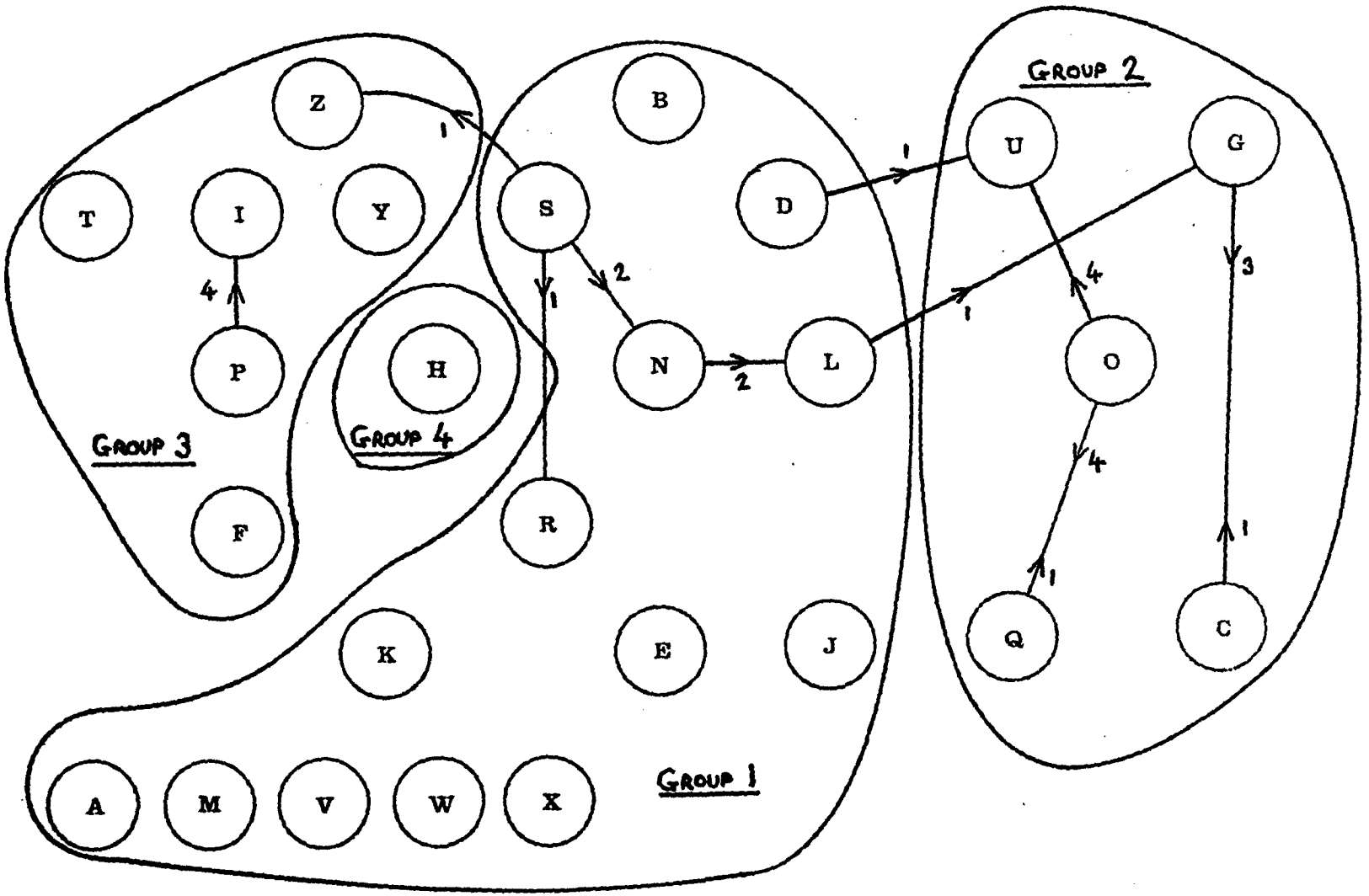
Group	Map
1	17
2	3

Fig. 5.6g Confusion Graph for Map P .



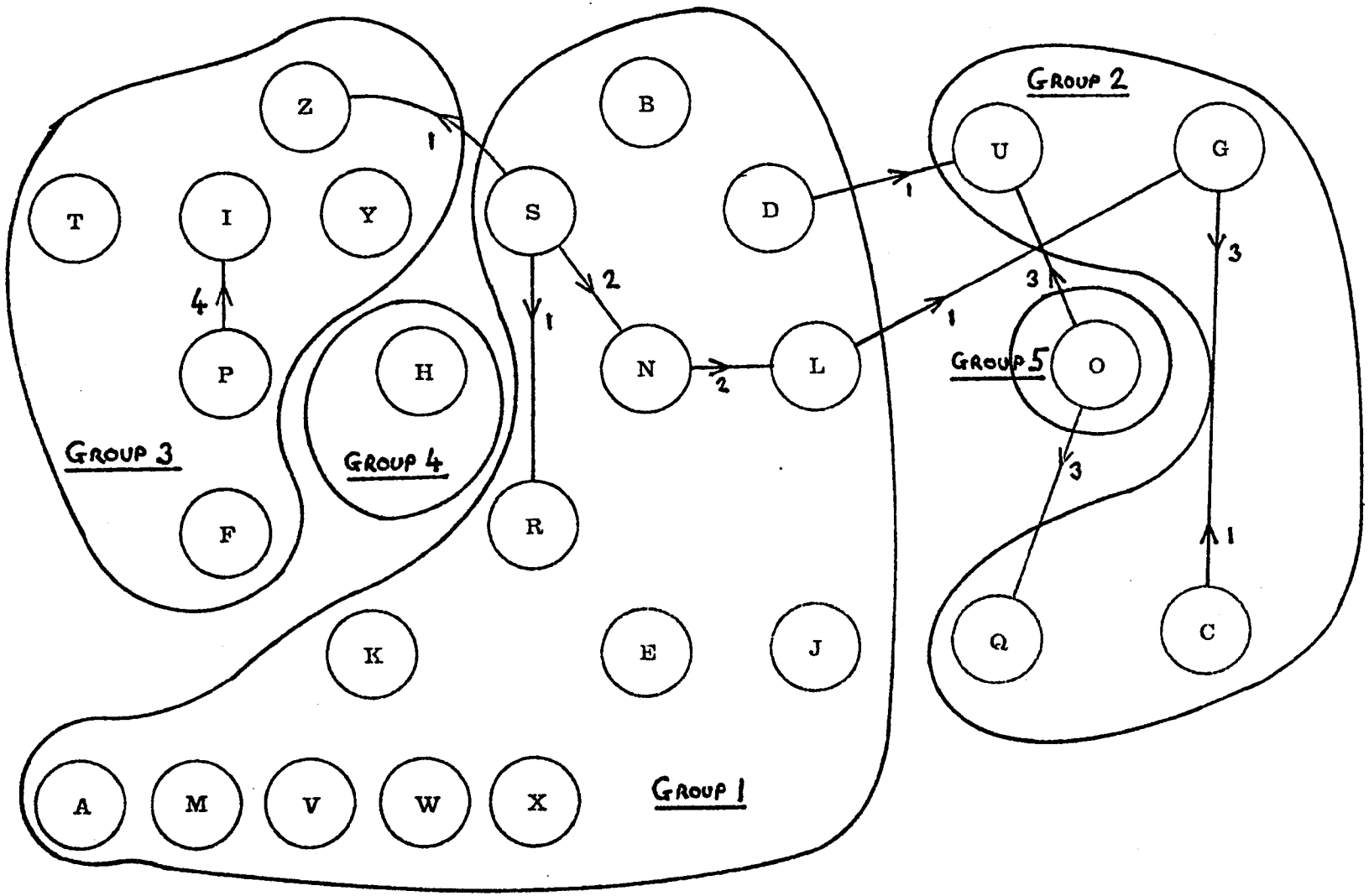
Group	Map
1	17
2	3
3	24

Fig. 5.6h Confusion Graph for Map Q .



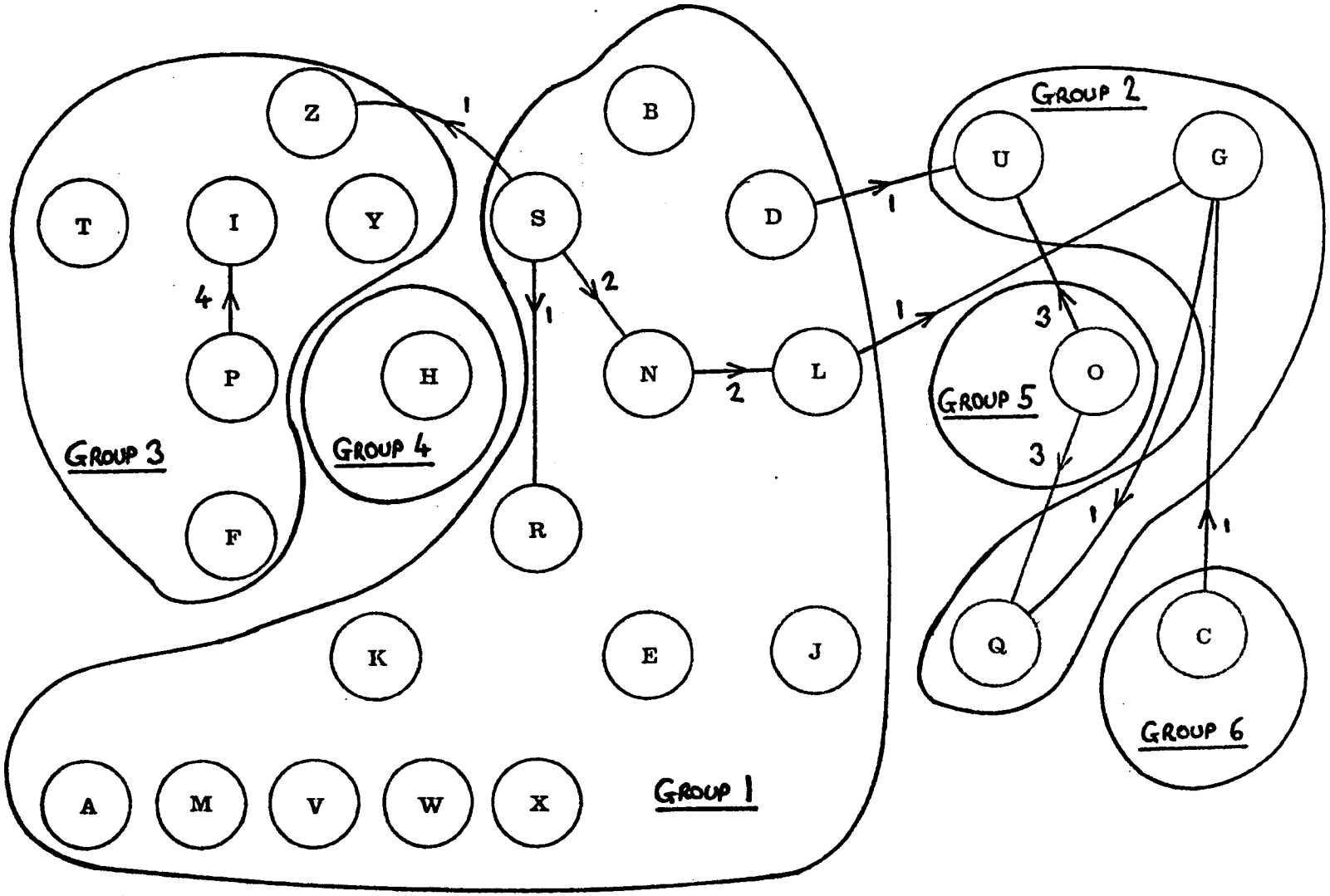
Group	Map
1	17
2	3
3	24
4	5

Fig. 5.61 Confusion Graph for Map R .



Group	Map
1	17
2	3
3	24
4	5
5	18

Fig. 5.6 j Confusion Graph for Map S .



Group	Map
1	17
2	3
3	24
4	5
5	18
6	27

Fig. 5.6k Confusion Graph for Map T .

in combining maps, but it is first necessary to establish which confusions are the most important so that they may be removed first.

Table 5.8 shows the twenty most commonly confused character pairs over 26 single random maps, using the same pattern sets as before. These twenty pairs account for over 80% of all the confusions, whilst the first seven pairs account for over 50%. Clearly from the maps in Fig 5.6 a - f, map 17 offers the best R/H performance (map 5 has fewer mis-classifications, but the R/N performance is extremely poor), whilst map 3 offers the best O/Q compromise. Maps 17 and 3 were thus combined to give map P, the confusion graph for which is in Fig.5.6g. Map 3 was chosen for the characters in group 2, namely C,G,O,Q,U, whilst Map 17 was used for group 1, namely all the remaining characters. This partition was chosen in order to keep the number of confusions within each group small, as well as minimizing the number of confusions between groups. A number of characters not confused in either of the single maps were assigned arbitrarily to group 1. All of the confusions in the combined map P arising within the groups 1 and 2 are exactly as in the individual maps, but the confusion D/U, between groups, was unexpected. Thus in combining the two maps the number of confusions has been reduced from 94 (map 3) or 74 (map 17) to 50 (map P) - a significant improvement. Table 5.9 shows this improvement in terms of recognition performance. The first row gives the performance over the range of single random maps 0 - 38 for both the optimising set and for a completely independent test set. The second row gives the performance of the combined map P.

Clearly there was no need to stop the optimisation process after just one pass, so map Q (Fig.5.6h) was derived from maps 17, 3, and 24, placing classes F, I, P, T, Y, Z in group 3 in order to improve on the

Table 5.8 Confused Character Pairs.

	Confused pair	No. of confusions	% of total confusions	Cumulative % of total confusions
1	R/H	483	14.9	14.9
2	Q/O	240	7.4	22.3
3	O/Q	229	7.1	29.3
4	O/U	216	6.7	36.0
5	P/I	196	6.0	42.0
6	S/U	185	5.7	47.7
7	R/N	164	5.1	52.8
8	S/N	131	4.0	56.8
9	P/F	109	3.4	60.2
10	C/G	106	3.3	63.5
11	S/Z	83	2.6	66.0
12	S/R	75	2.3	68.3
13	R/K	67	2.1	70.4
14	S/Q	67	2.1	72.5
15	P/R	60	1.8	74.3
16	S/H	59	1.8	76.1
17	Q/G	57	1.8	77.9
18	S/B	55	1.7	79.6
19	R/U	52	1.6	81.2
20	Q/U	43	1.3	82.5

Total confusions : 3244
Total confused pairs : 71

Font (d) UC, 12x16, n=8. Train: 400/class. Test: 100/class.
Maps 0-25. Classes A-Z.

Table 5.9 Recognition Performance of Combination Maps.

Map Name	No. of groups	% correct classifications	
		Optimising set	Test set
0 - 38	1	97.6 (best) 95.8 (mean)	97.5 (best) 96.6 (mean)
P Fig. 5.6 g	2	98.4	97.8
Q Fig. 5.6 h	3	98.8	97.9
R Fig. 5.6 i	4	99.1	97.9
S Fig. 5.6 j	5	99.2	97.8
T Fig. 5.6 k	6	99.3	97.8

Font (d) UC, 12x16, n=8. Train: 400/class. Test: 200/class. A-Z.
Optimising set: 100/class.

T/I, P/I, P/F, P/Y performance of map P. Less expected was the improvement in S/Z, S/I, S/Y performance between groups 1 and 3. Again the overall improvement can be seen in Table 5.9. This process was repeated several times, until in map T there were six groups, some containing only one character.

The merit of this optimising procedure lies in its ability to combine the discriminatory powers of different groups of features (n-tuples) without having to identify the features themselves. Clearly one requires a large number of individual random maps from which to build the combined maps: those in Fig.5.6 were chosen from a total of 39 maps, of which only about 6 were found to be of great use.

Care must be exercised at each stage of the optimising process since one has no guarantee that in reducing confusions within a group other confusions will not arise between the groups, or even that within-groups confusions will be reduced to the level exhibited in the corresponding individual (single-group) map. Since the effectiveness of each stage must be assessed on an empirical basis, it is important that an independent check be made using a separate test set. In Table 5.9 the test set figures do not improve as markedly as the optimising set performance figures: this indicates that the two sets are not sufficiently representative of the patterns.

It should be noted that the improvements obtained using this optimising scheme did not increase the storage requirements of the recognizer at all: there were merely re-adjustments in the connections between the pattern matrix and the n-tuple (RAM) inputs.

5.4.4 Combinations of Random Sub-maps

It was shown above (Section 5.4.2) that the performance figures (in terms of correct classifications) of random mappings cover a very narrow range, so that the probability of finding a mapping which exhibits much better than average performance is very small. However, Section 5.4.1, and particularly Fig.5.3 showed that the use of groups of n-tuples smaller than complete mappings leads to greater fluctuations in the recognition performance, although the actual performance levels are lower. The fluctuations arise because in a small group the influence of individual features (n-tuples) is greater and thus the presence of an outstandingly 'good' feature or set of features is reflected more directly in the performance figures. Large collections of n-tuples such as complete mappings tend to average out the effects of individual n-tuples, giving only a small spread of performance figures. Having identified useful groups of n-tuples (referred to here as 'sub-maps' since they are subsets of full random maps) one may recombine them into full maps with the aim of improving the performance. It is not obvious which is the optimal size of sub-map, or indeed whether there is an optimal size. Furthermore the extent to which the individual sub-maps interact (constructively or destructively) when recombined into full maps is not known.

An experiment was conducted in order to establish whether it is possible to improve on the performance of random full maps by the selection and recombination of sub-maps. Patterns from font (d) UC classes C,G,O,Q and U were used, with dimensions 12x16, and n-tuple size of 8. The restriction to only five classes was imposed in order to allow a sufficiently large number of test runs to be made in the time available. The classes chosen had proved particularly troublesome in earlier tests.

A total of 39 random maps was tested, firstly using the full maps (24 n-tuples per sub-map); secondly using 75 sub-maps of 12 n-tuples each; thirdly using 117 sub-maps of 8 n-tuples each; fourthly using 133 sub-maps of 6 n-tuples each; and lastly using 180 sub-maps of 3 n-tuples each. The same training and testing (really 'optimising') sets were used for every sub-map, and the performance distribution histograms are given in Fig.5.7. Two features are immediately obvious: firstly the mean performance for a particular size of sub-map is dependent upon the size, and secondly the smaller sub-maps exhibit a much greater range of performance figures than the larger sub-maps.

For each sub-map size, selected sub-maps were recombined into ten separate full maps covering a range of values of the mean performance of the constituent sub-maps. For example the first test used a full map consisting of the best 8 sub-maps of 3 n-tuples each, having the individual performance figures 68.8%, 65.3%, 62.7%, 62.4%, 60.5% 60.2%, 59.5% and 58.3%, the mean of which is 62.2%. The performance achieved by this recombined map on the same test set as previously was 97.6%. Thus in all 40 full maps were tested (obviously the case of 24 n-tuples per sub-map could not be used since these were already full maps).

In Fig.5.8 the performance of each recombined full map is plotted against the mean performance of its constituent sub-maps for groups of 3, 6, 8 and 12 n-tuples per sub-map (i.e., 8, 4, 3 and 2 sub-maps per full map respectively). In each case the curve has a generally positive gradient, showing that there is a correlation between the mean performance of a group of sub-maps and the overall performance of the group when recombined into a full map. Furthermore the upper points of each curve lie well above the line representing the mean performance of the random full maps, and most of the upper points lie above the line representing the best performance of any of the

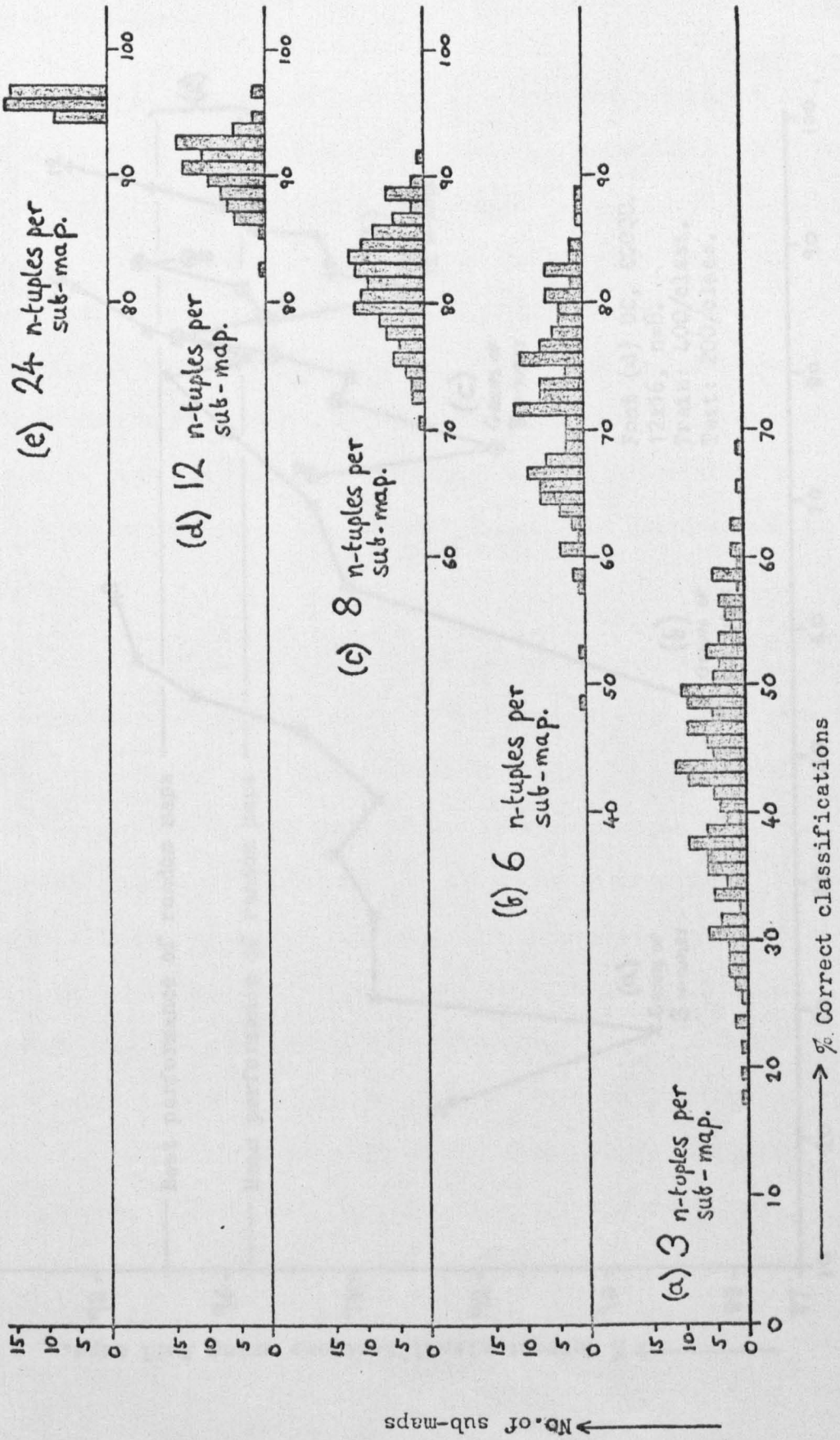


Fig. 5.7 Performance Distribution of Sub-maps.

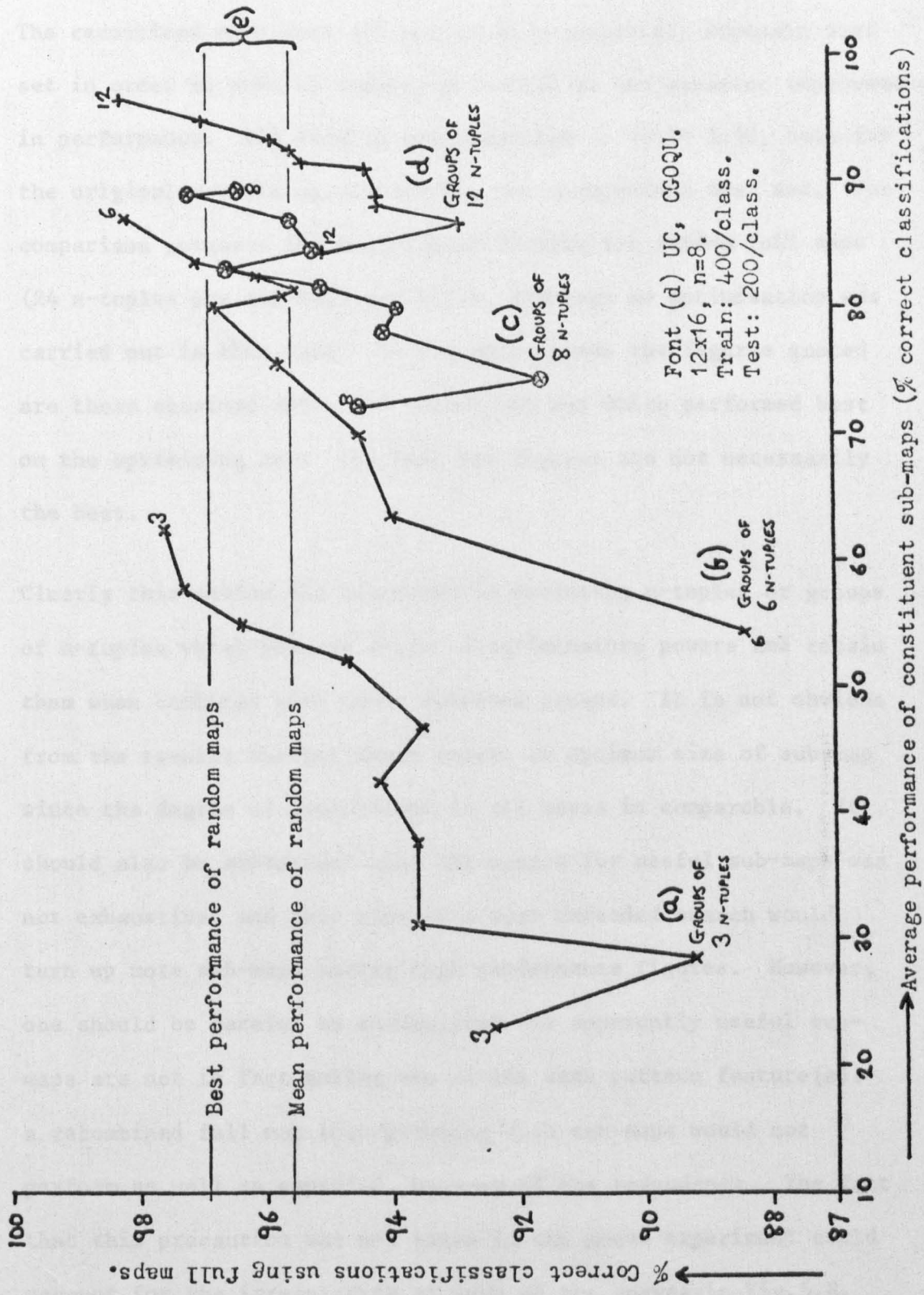


Fig. 5.8 Performance of Recombined Full Maps.

random full maps tested.

The recombined maps were all tested on a completely separate test set in order to make an independent check on the apparent improvement in performance. The results are presented in Table 5.10, both for the original optimising set and for the independent test set. For comparison purposes the performance figures for random full maps (24 n-tuples per sub-map) are given, although no optimisation was carried out in this case. In all other cases the figures quoted are those obtained using the recombined map which performed best on the optimising set: the test set figures are not necessarily the best.

Clearly this method has succeeded in isolating n-tuples or groups of n-tuples which possess useful discriminatory powers and retain them when combined with other selected groups. It is not obvious from the results whether there exists an optimum size of sub-map since the degree of improvement in all cases is comparable. It should also be emphasised that the search for useful sub-maps was not exhaustive, and that clearly a more extended search would turn up more sub-maps having high performance figures. However, one should be careful to ensure that two apparently useful sub-maps are not in fact making use of the same pattern feature(s): a recombined full map incorporating such sub-maps would not perform as well as expected, because of the redundancy. The fact that this precaution was not taken in the above experiment could account for the irregularity of some of the curves in Fig.5.8.

Table 5.10 Performance of Recombined Full Maps.

No. of n-tuples per sub-map	Best performance of recombined maps (% correct classifications)	
	Optimising set	Independent test set
3	97.6	98.8
6	98.2	98.5
8	97.2	97.1
12	98.3	99.3
24 *	96.9 (map no. 25)	98.7 (map no. 34)

All font (d) UC. 12x16, n=8, classes C,G,O,Q,U.
Train: 400/class. Test: 200/class. Optimise: 200/class.

* For comparison only -- no optimisation carried out.

5.4.5 Selection of n-tuples by Number of Rejections

The experiment described in this section is concerned with the generation of new mappings by combining n-tuples selected from a number of random maps on the basis of measurements made on the individual n-tuples. The criterion used in making the selection is referred to as the number of 'n-tuple rejections' since it is an inverse measure of the discriminatory power of an n-tuple on a particular set of patterns, and is based on the occurrence frequencies of the n-tuple states. The idea of creating new maps based on single n-tuple measurements is really an extension of the idea presented in the previous section, using sub-maps consisting of just one n-tuple. The measurement cannot of course be the number of correct classifications in this case, because an isolated n-tuple does not possess the power of generalisation necessary for pattern recognition: it responds only to those patterns seen during training.

Firstly it is necessary to explain how the state occurrence frequencies of each candidate n-tuple were measured, and this is best done with reference to Table 5.11 which was derived for 400 patterns of each of the classes C, G, O, Q and U from font (d) UC. The first line after the title and n-tuple number shows that 9 different states were seen in patterns of class C; the state 2 octal (i.e.,00 000 010 binary) was seen in 22 patterns, the state 40 (i.e., 00 100 000) was seen in 112 patterns, and so on. The states seen in patterns of the other classes are presented in a similar manner, together with the total number of states seen (counting each class separately) and the number of different states seen (counting all classes together). The fact that these two values are different indicates that some states were seen in patterns of more than one class. These are enumerated in the next section of the table, for example the state 0

E55B.MMP map 0. n=8, 12x16.
N-tuple no. 0

C	9	2:	22	40:	112	42:	98	44:	63	46:	10	50:	6	52:	30	
	64:	52	66:	7												
G	14	0:	5	2:	51	10:	1	12:	6	40:	52	42:	34	44:	67	
	46:	14	50:	31	52:	107	64:	23	110:	1	112:	1	150:	7		
O	11	2:	17	12:	35	24:	3	42:	35	44:	13	50:	89	52:	116	
	54:	19	64:	32	70:	4	74:	37								
Q	11	4:	1	12:	10	14:	4	40:	9	42:	52	44:	69	50:	68	
	52:	160	54:	14	64:	12	74:	1								
U	14	0:	22	2:	2	10:	4	12:	70	40:	23	42:	14	44:	39	
	46:	3	50:	21	52:	141	54:	3	56:	1	60:	2	64:	55		

Total states: 59 Different states: 23

Multiple states.

0/ 2:	5	22			
2/ 4:	22	51	17	2	
10/ 2:	1	4			
12/ 4:	6	35	10	70	
40/ 4:	112	52	9	23	
42/ 5:	98	34	35	52	14
44/ 5:	63	67	13	69	39
46/ 3:	10	14	3		
50/ 5:	6	31	89	68	21
52/ 5:	30	107	116	160	141
54/ 3:	19	14	3		
64/ 5:	52	23	32	12	55
74/ 2:	37	1			

Multiple states: 13 Rejections: 1969 = 98.4%

Table 5.11 State Occurrence Frequencies for One n-tuple.

occurred in two different classes, with frequencies 5 and 22. Reference to the first part of the table shows that these were for the classes G and U respectively. States occurring in patterns of more than one class are referred to as ' multiple states', of which there were 13 in this case. The total number of patterns giving rise to one of these multiple states is referred to as the number of rejections, and is 1969 in this case, representing 98.4% of the 2000 patterns in the set.

The word 'rejection' is used because whenever one of the multiple states occurs in a test pattern, at least two discriminators respond positively, thus giving less discriminatory information than in the case of a non-multiple state, where only one disciminator responds. The existence of multiple states therefore increases the likelihood that a pattern will be rejected by the recognizer, and is thus undesirable.

Table 5.12 is a summary of the n-tuple occurrence frequency figures for all 24 n-tuples of a particular mapping (the first n-tuple is the same as in Table 5.11), and shows that the number of rejections ranges from 98.4% to 96.6%, and that the mean value for the whole mapping is 97.6%. The mapping used for Table 5.12 was not in fact random, but was composed of the 24 worst n-tuples (worst in the sense of having the most single n-tuple rejections) from 40 different random maps - a total of 960 n-tuples. For the purposes of the experiment these 960 n-tuples were re-configured into 40 new non-random maps exhibiting the widest possible range of mean percent single n-tuple rejections. The 'best' map constructed in this way had a mean rejection value of 66.8%.

Each of these new maps was used in training the recognizer (using the same set of patterns as for the frequency measurement process),

E55B map 0. n=8, 12x16.

N-tuple no.	Total	Diff.	Mult	Rejects	
0	59	23	13	1969	98.4 %
1	99	37	28	1968	98.4 %
2	83	31	23	1967	98.3 %
3	98	39	28	1966	98.3 %
4	126	47	37	1965	98.3 %
5	93	36	22	1964	98.2 %
6	76	28	20	1961	98.1 %
7	67	25	18	1962	98.1 %
8	80	30	18	1961	98.1 %
9	58	20	15	1960	98.0 %
10	72	26	17	1959	97.9 %
11	105	35	29	1955	97.8 %
12	109	44	27	1955	97.8 %
13	90	33	24	1953	97.7 %
14	74	34	18	1946	97.3 %
15	138	58	43	1945	97.3 %
16	100	40	28	1940	97.0 %
17	116	47	28	1938	96.9 %
18	91	39	26	1933	96.7 %
19	73	27	19	1933	96.7 %
20	99	40	28	1934	96.7 %
21	156	67	42	1933	96.7 %
22	85	31	24	1931	96.6 %
23	142	63	47	1931	96.6 %
Averages :	95.4	37.5	25.9		97.6 %

Table 5.12 State Occurrence Summary for Complete Mapping.

and in testing it on two separate sets of patterns (Test Sets 1 and 2). The whole procedure was repeated for a further 40 random maps whose n-tuples were re-configured into 40 maps showing a wide range of the mean percent single n-tuple rejections. The results for the total of 80 re-configured maps are presented in Fig.5.9 for Test Set 1 and in Fig.5.10 for Test Set 2. Also indicated on the scatter diagrams are the best and worst performance figures obtained using the original random maps. Each plotted point in the diagrams represents one re-configured mapping.

Clearly mappings having a large mean rejection figure perform least well, having about five times the error rate of any of the purely random maps. As the number of rejections decreases, the performance improves, but only up to the mean level of the purely random maps: there is no significant improvement on this even for the mappings having the least rejections. The results show that in choosing n-tuples for a mapping one should avoid those having a large number of rejections, but that there is little point in selecting only those n-tuples having a small number of rejections.

The lack of success in using this method to improve recognition performance may be attributed to two factors.
Firstly one has no assurance that the set of patterns on which the measurements are made (the training set) is adequately representative of the test patterns (although previous optimisation experiments on the same data, reported above, have shown that improvement is possible). If the multiple states occurring in the test set have very different occurrence frequencies to the same states in the training set, one loses the benefit of the optimisation. The second contributory factor is that the measurements are made on single n-tuples: no account has been taken of the way in which

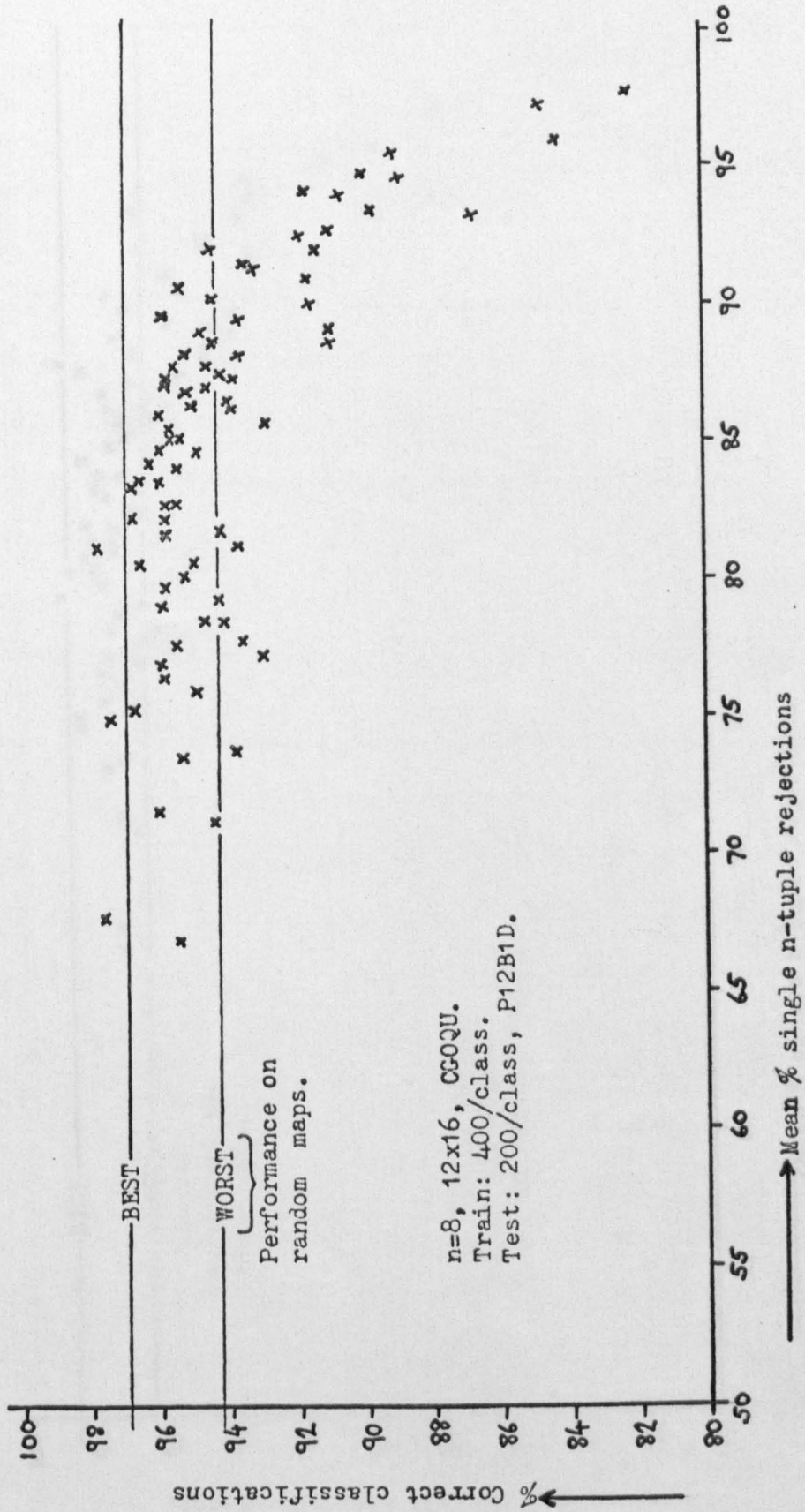


Fig. 5.9 Performance on Test Set 1.

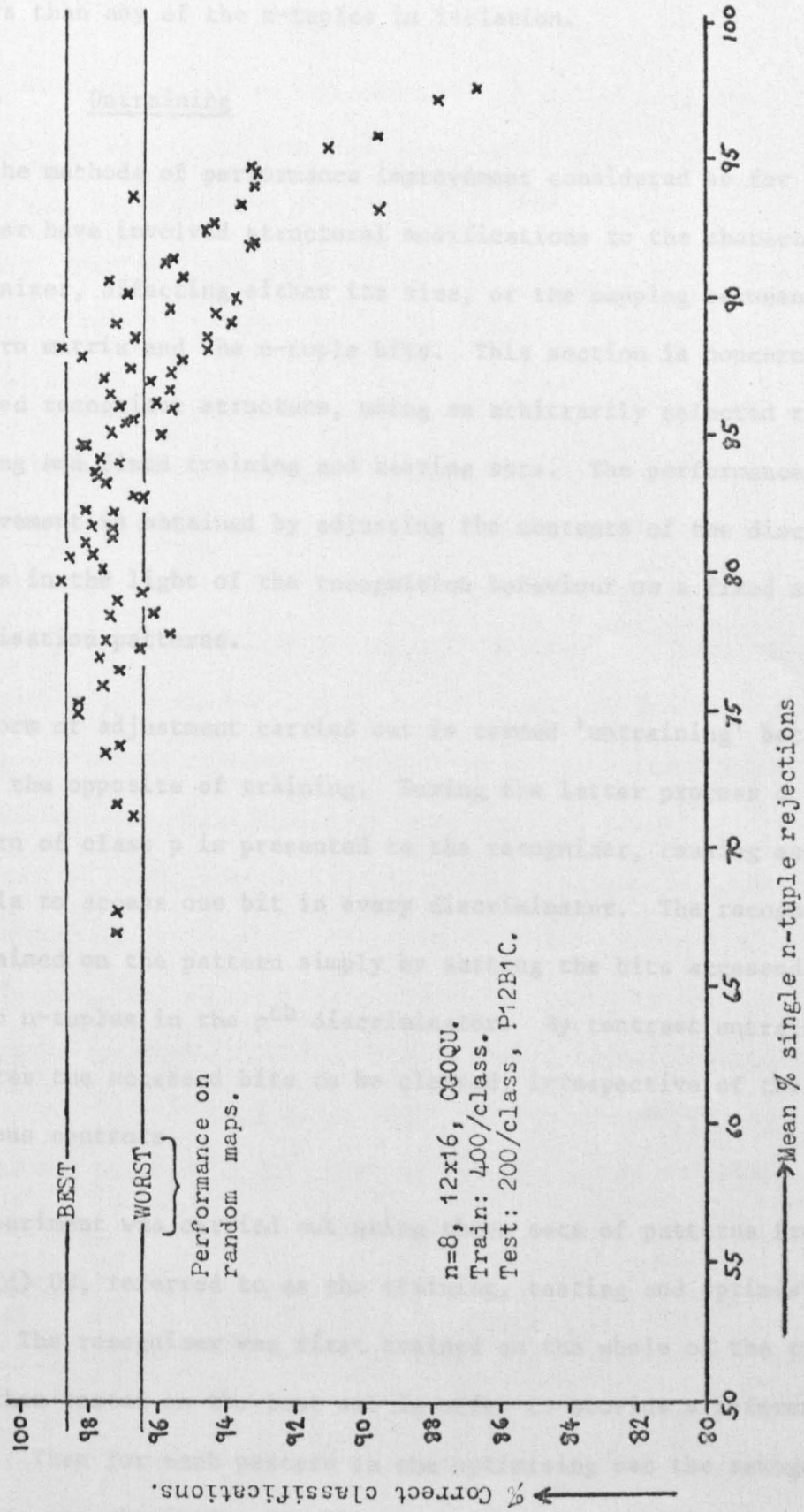


Fig. 5.10 Performance on Test Set 2.

several n-tuples may act as a group, having greater discriminatory powers than any of the n-tuples in isolation.

5.4.6 Untraining

All the methods of performance improvement considered so far in this chapter have involved structural modifications to the character recognizer, affecting either its size, or the mapping between the pattern matrix and the n-tuple bits. This section is concerned with a fixed recognizer structure, using an arbitrarily selected random mapping and fixed training and testing sets. The performance improvement is obtained by adjusting the contents of the discriminator stores in the light of the recognition behaviour on a fixed set of optimisation patterns.

The form of adjustment carried out is termed 'untraining' because it is the opposite of training. During the latter process a training pattern of class p is presented to the recognizer, causing each n-tuple to access one bit in every discriminator. The recognizer is trained on the pattern simply by setting the bits accessed by the n-tuples in the p^{th} discriminator. By contrast untraining requires the accessed bits to be cleared, irrespective of their previous contents.

An experiment was carried out using three sets of patterns from font (d) UC, referred to as the training, testing and optimising sets. The recognizer was first trained on the whole of the training set, then tested on the test set in order to provide a reference point. Then for each pattern in the optimising set the recognizer's response was obtained. If this was a correct classification or a rejection, no further action was taken, and the next pattern tested. If the response was incorrect, however, the discriminator giving (erroneously) the maximum response was untrained on the pattern

before proceeding. At the end of this first pass the number of patterns used for untraining was recorded, and the recognition performance re-tested on the test set. A second pass was then made through the optimising set, and so on until eventually none of the optimising patterns were mis-classified. The whole process was repeated several times using different testing threshold values in the optimising stage. No threshold was used for the performance testing stage.

Fig.5.11 shows the classification performance plotted against the total number of untraining patterns for each of the thresholds used. The choice of threshold is significant because it determines what degree of mis-classification can be tolerated. The general degradation in performance is a direct result of the clumsiness of the untraining approach: untraining on one pattern makes large inroads into the n-tuple stores, unselectively removing both useful and harmful 'seen' states. The fact that a slight improvement in performance is obtained in two cases shows that untraining can be advantageous, but clearly its use must be carefully restricted, and its effects monitored. The overall downward trend in performance is because most of the confusions occur between similar pattern classes. For example untraining the 'Q' discriminator on an 'O' removes many of the states characteristic of Q's, because of the similarity of the characters. This accounts for the fact that relative thresholds perform better than absolute thresholds for untraining purposes: misclassifications occurring with a large relative threshold in operation must be due to patterns that are fairly unrepresentative of their class because the correct discriminator is responding much less strongly than some other spurious discriminator - untraining on these patterns could be beneficial. On the other hand, misclassifications occurring when there is a large absolute threshold in effect

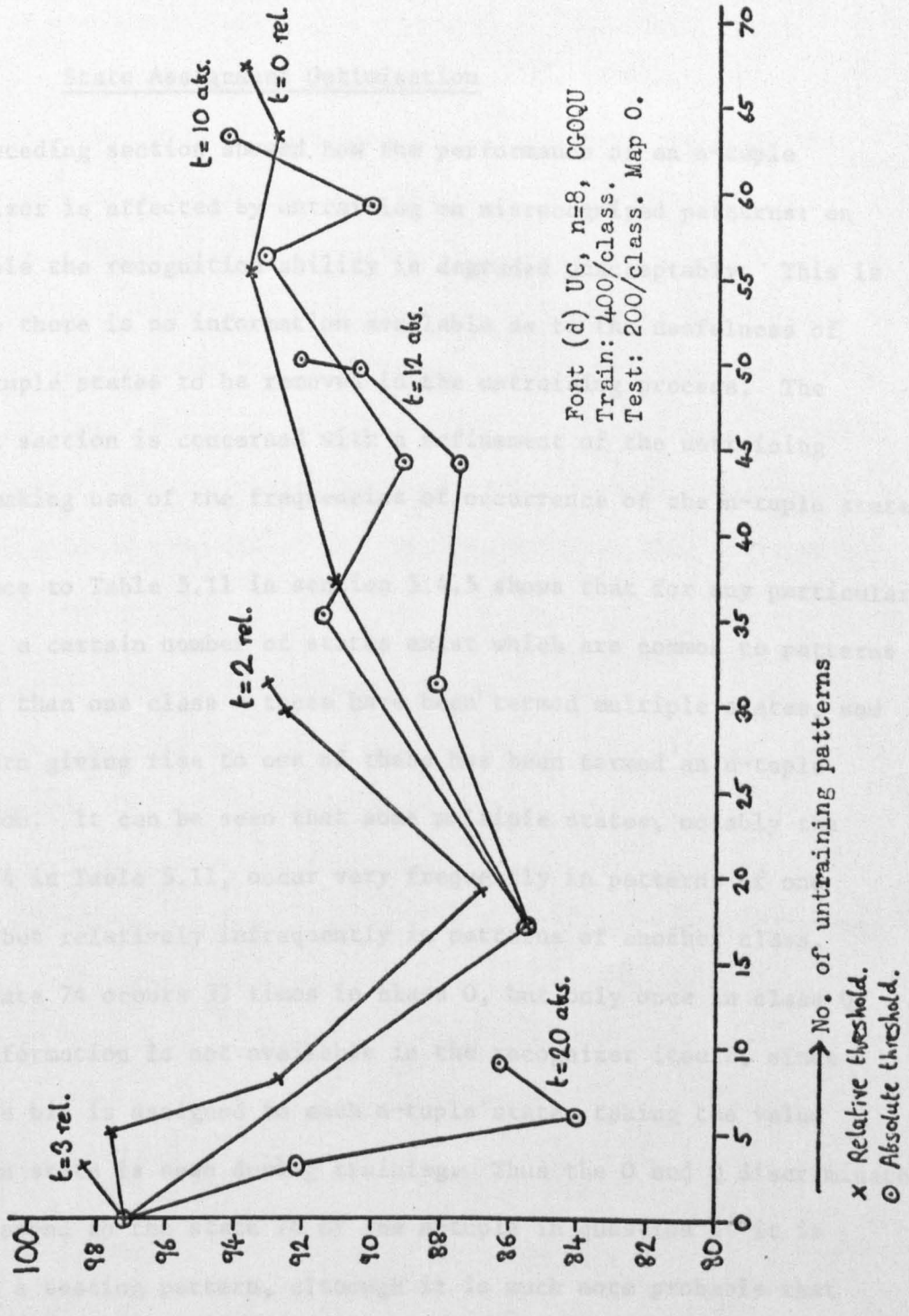


Fig. 5.11 Untraining Behaviour for Various Thresholds.

are due to patterns very similar to those of the class whose discriminator is responding most strongly, since they have a large number of states in common - untraining on such patterns would be largely detrimental.

5.4.7 State Assignment Optimisation

The preceding section showed how the performance of an n-tuple recognizer is affected by untraining on misrecognized patterns: on the whole the recognition ability is degraded unacceptably. This is because there is no information available as to the usefulness of the n-tuple states to be removed in the untraining process. The present section is concerned with a refinement of the untraining idea, making use of the frequencies of occurrence of the n-tuple states.

Reference to Table 5.11 in section 5.4.5 shows that for any particular n-tuple a certain number of states exist which are common to patterns of more than one class - these have been termed multiple states, and a pattern giving rise to one of these has been termed an n-tuple rejection. It can be seen that some multiple states, notably the state 74 in Table 5.11, occur very frequently in patterns of one class, but relatively infrequently in patterns of another class. Thus state 74 occurs 37 times in class O, but only once in class Q. This information is not available in the recognizer itself, since only one bit is assigned to each n-tuple state, taking the value o if the state is seen during training. Thus the O and Q discriminators both respond to the state 74 of the n-tuple in question if it is seen in a testing pattern, although it is much more probable that the pattern belongs to class O than to class Q. This situation could be improved by deleting the state 74 from the n-tuple in the Q discriminator, thus decreasing the original number of n-tuple rejections by 37 (since this would no longer be a multiple state),

but at the cost of one deliberate mis-classification by the n-tuple.

This then was the basis of an experiment in which four levels of 'pruning' of the multiple states were used in order to reduce the number of n-tuple rejections, at the cost of increased n-tuple mis-classifications. Table 5.13 shows how the recognition performance changed with each pass through the state reassignment procedure: the initial condition was prior to any pruning, and passes i), ii) and iii) were after progressively more severe pruning of the multiple states. Tests were made on two maps: map 0 being the worst performer in the experiments of section 5.4.5, and map 13 being the best. The mean percent n-tuple mis-classifications refers to the proportion of training patterns giving rise to states which had been removed from the n-tuple stores, and were thus assigned to some other class.

Inspection of Table 5.13 shows that there is very little change in performance even for very heavy pruning of multiple states. Even when approximately half of the training patterns give rise to re-assigned states (i.e., are mis-classified by individual n-tuples) the performance is only slightly worse than its initial level. The fact that the performance does not improve as the mean percent. n-tuple rejections decreases is further evidence that making measurements on isolated n-tuples is inadequate for the purpose of optimising the recognition ability.

5.5 Summary

The first part of the chapter was concerned with experiments on upper-case (UC), lower-case (LC), and multifont recognition. Only a small sample of the many available fonts was used, but the principle has been demonstrated: that one can use the same discriminators for

Table 5.13 State Assignment Optimisation.

	Map 0				Map 13			
	Initial state	Pass (i)	Pass (ii)	Pass (iii)	Initial state	Pass (i)	Pass (ii)	Pass (iii)
Mean % n-tuple mis-classifications	0.0	6.8	14.6	44.7	0.0	7.4	14.9	48.7
Mean % n-tuple rejections	97.6	94.9	92.6	89.3	88.1	83.7	81.3	77.3
% Correct classifications	86.7	86.2	88.3	85.7	98.8	98.4	98.5	98.5

Font (d) UC. n=8, 12x16, CGOQU. Train; 400/class. Test: 200/class from P12B1C.

fairly similar fonts, possibly switching to a different set for a very different font or size of type. Accommodation of UC and LC in the same discriminators also appears feasible, but a more reliable arrangement would be two sets of discriminators, particularly where each is trained on several fonts. Alternatively one may employ an n-tuple size large enough to accommodate all the pattern variations desired, i.e., font, size, and case. For adequate LC recognition, however, it would be necessary to employ greater camera resolution than was used in the experimentation.

Experiments on tracking show that adequate performance can be attained given a high enough camera frame rate. In a hardware implementation the cycle time of the recognizer would be sufficiently small to allow much more sophisticated forms of automatic centring and tracking: the algorithm used here has very obvious limitations. For example, extension of the 'search' function horizontally as well as vertically would provide much more accurately centred characters and thus improve the recognition performance. The segmentation experiments were somewhat inconclusive in that the performance levels attained would barely be sufficient for the intended application. This is considered to be due to two factors: firstly the camera resolution was on some occasions insufficient, resulting in some LC characters being indistinguishable, and secondly the crude vertical tracking algorithm was at times misled by the characters adjacent to the one at the centre of the field of view, particularly where there was a marked difference in the heights of the characters.

The main bulk of the chapter was concerned with optimisation experiments, i.e. attempts to improve the recognition performance. Experiments on varying the number of n-tuples show that it is not sufficient to use a 1:1 mapping merely because it is convenient: one should consider

alternative configurations. The most successful optimisation experiment involved the use of combinations of random maps in order to make the greatest possible use of the discriminatory characteristics of the different maps. This technique appears to be particularly useful because it offers a direct means of correcting classification errors without any increase in storage size. The only cost is a slight increase in the complexity of the connections between the input matrix and the discriminators.

Encouraging results were also obtained by using performance measurements on 'sub-maps' - small groups of n-tuples - to create new mappings which perform significantly better than the original random mappings, again with no increase in recognizer size. Two sets of experiments based on the frequencies of occurrence of n-tuple states were largely unsuccessful, mainly because they relied on the assumption that each n-tuple acts independently, which is clearly invalid. The use of frequency measurements on groups of n-tuples would be expected to yield better results, particularly in view of the superior performance of the frequency memory matrix as a recognizer. Experiments on 'untraining' did not prove very encouraging because of the gross insensitivity of the storage modification procedures.

Two important and not entirely unexpected conclusions have emerged from the optimisation experiments: firstly that all the patterns used must be adequately representative, and secondly that the measurements must be made on groups of n-tuples rather than n-tuples in isolation. Finally it should be noted that the optimisation techniques presented here may be used either singly or in combination, for example there is no reason why the re-combined maps created from selected sub-maps (section 5.4.4) should not themselves be selected according to their inter-class discriminatory properties (section 5.4.3).

Chapter 6

Conclusion

6.1 Summary and Discussion of the Thesis

The aim of the research reported in this thesis was to establish the feasibility or otherwise of using an n-tuple character recognizer in a reading aid for the blind. Chapter 1 showed the need for a device offering more than direct translation of the printed image, i.e., one which is able to recognize the text and pass on the recognition decisions to the blind user in a readily assimilated form, such as synthetic spelled speech. The form of such a device was outlined, with a list of the attributes required of its character recognizer, namely, fast input rate, fast recognition, tolerance of multiple fonts, ability to accommodate the tracking errors inherent in input from a hand-held probe, low error rate, and low cost.

The n-tuple method of pattern recognition was introduced in Chapter 2, with a detailed review of previous work in the field. The method was chosen largely because of its versatility: its implementation in hardware is very easily achieved (this is discussed below, in section 6.2). Clearly there are alternative methods which would yield comparable recognition performance, but without the directness and flexibility of the n-tuple method: a microprocessor-based implementation of some algorithmic method would not be capable of the high speed achieved by a ROM or PLA-based n-tuple recognizer, whilst the development of specialised hardware for other methods would incur a greater cost penalty. In both cases one must bear in mind that a successful reading aid for the blind must be both portable, and thus compact, and cheap enough to be produced on a large scale.

The first part of Chapter 3 was a discussion of the specific character recognition problems to be encountered, in terms of the

variability of the source patterns due to variations in style (font) and quality of the printed image, random variations due to electrical noise in the camera system, distortion of characters due to the finite resolution of the digitizing matrix, positional variations in the characters as the text is scanned, errors in the focussing and illumination of the text, and adjacency effects due to the nature of continuous print. The remainder of Chapter 3 was concerned with details of the structure of the n-tuple recognizer, and their effect on recognition ability. The parameters concerned are n-tuple size, pattern dimensions, number of n-tuples, number of classes, mappings, and decision logic. Finally the importance of the correct choice of patterns for training, testing and optimising was discussed.

Chapter 4 gave detailed descriptions of the solid-state camera system which was specially developed for this work, and of the extensive set of computer programs necessary for simulating the various character recognizer structures employed in the research and for developing various optimisation techniques. Although these programs, and in fact the entire subsystem known as JANSYS (described in detail in Appendix B), were written specifically for the research in hand, their modular structure makes them easily adaptable to any pattern recognition task using the n-tuple technique. The addition of new commands can be effected quite simply by the insertion of a subroutine which may or may not call other subroutines within the system, and so one may build up a hierarchy of functions, all of which are directly accessible to the user, via the console, but at different levels of complexity. Changes in the structure of the simulated recognizers may again be effected simply by replacing the appropriate subroutine.

A full description of the experiments was presented in Chapter 5. These were set out in three parts, firstly those demonstrating the

ability of the n-tuple recognizer to accommodate upper-case, lower-case and multifont characters. The second series of experiments established that the character recognizer in question was able to handle errors in the position of the characters within the field of view, and, to a more limited extent, was able to segment running printed text into its constituent characters. Finally a number of experiments on optimisation was presented. Those which depended upon measurements of large-scale properties of the recognizer appeared to be the most successful.

These included the choice of the number of n-tuples, the use of combinations of random maps specially chosen to minimize particular character confusions, and the re-combination of small groups of n-tuples (termed sub-maps) selected according to performance criteria. Further experiments based on single n-tuple measurements were less successful in spite of using information concerning the n-tuple state occurrence frequencies which was not available to the other optimisation schemes. The method of 'untraining' was found to be of limited value because of its gross effects on the storage contents of the recognizer.

The remainder of this chapter is concerned with suggesting ways in which the character recognizer could be implemented for a blind reading aid, and in outlining those areas of n-tuple character recognition considered to require further research.

6.2 Implementation

An outline of the complete reading aid is given in Fig.1.1 (section 1.3). The image of the printed text obtained by the camera is presented to the character recognizer which determines whether a recognizable character is visible near the centre of the

field of view and if so to which class it belongs. This decision is passed to the utterance generator which transmits it to the blind user in an audible form. By moving the camera input probe along the line of text the user is able to 'read' letter by letter. To prevent gross mistracking on the part of the user, the device emits warning signals whenever the probe appears to be drifting away from the line of text, and also at the end of the line.

A solid-state device would appear to be the most suitable for the camera, because of its several distinct advantages over more conventional (vacuum-tube) devices, viz. compactness, ruggedness, modest power requirements, high frame rate, and high sensitivity. Furthermore such cameras generally require less complex signal processing circuitry. The work reported in this thesis made use of data from a 64 x 64 bit photodiode array, but there is no real reason why a charged-coupled device (CCD) photomatrix should not be used in an implementation of the reading aid, although the requirement of 50 - 100 frames per second might be difficult to achieve without undue image distortion (for example, 'smear'). However, the large size of currently available CCD arrays (of the order of 256 x 256 elements) would certainly obviate any resolution problems.

The hand-held probe would be simply a light-pipe consisting of sufficient optical fibres to produce a bright image of the text at the camera (i.e. on the photomatrix). Illumination of the text would be achieved by connecting some of the fibres in the light-pipe to a light source.

The basic structure of the n-tuple character recognizer is shown in Fig.4.2 (section 4.4), or Fig.4.5 if the form of map optimisation discussed in section 5.4.3 is adopted. The number of discriminators

depends largely on the flexibility required of the system. Clearly the minimum necessary is 26, one for each letter of the alphabet, each being used for both upper-case and lower-case letters (as in section 5.2.2) and each being responsive to letters of several different fonts (as in section 5.2.4). To accommodate type fonts varying more widely in size and style it would be necessary to use one or more further banks of discriminators which are manually switched in as appropriate. To achieve greater reliability of recognition one could double the number of discriminators in order to handle upper-case and lower-case letters separately, thus allowing one to use two levels of resolution for the two types of character whilst retaining the same number of final pattern elements. Thus the upper-case discriminators would 'see' a low resolution image of a large area (equal to the size of an upper-case character) whilst the lower-case discriminators would 'see' a high resolution image of a smaller area (equal to the size of a lower-case character). This would solve the problems encountered in lower-case recognition in Section 5.2. Furthermore one could increase the number of fonts handled by one set of discriminators because in keeping upper- and lower-case letters separate one considerably reduces the amount of pattern variability within classes in the training set.

Accommodation of tracking errors on the part of the user would be achieved by allowing the pattern matrix (or matrices in the case of the dual resolution system) - the input to the recognizer - to be movable within the much larger frame received from the camera (say 64 x 64 bits, i.e., 4k). Thus by means of a suitable algorithm the 'window' would be able to track any deviations of the incoming line of text, and emit warning signals to the user when the limits of tracking are approached. The number of such window movements allowed for each camera frame is dependent both on the frame rate

and on the time required to extract the window(s) from the RAM containing the complete frame.

Suppose a window of 32 x 32 = 1024 bits is extracted from the camera frame. The lower-case discriminators 'see' a 16 x 16 window taken from the centre of the larger area, but at the same resolution, whilst the upper-case discriminators 'see' a 16 x 16 image of the whole area, but at reduced resolution. Thus 1024 bits have to be read from the 4k RAM holding the whole frame, giving a total access time of the order of 50 microseconds (assuming a RAM cycle time of 200 ns and four bits per memory word). A camera frame rate of between 50 and 100 per second means that between 200 and 400 window movements are permitted per frame. This is ample to allow each letter to be centred with a fair degree of accuracy, using both vertical and horizontal movements of the window. A block diagram of the suggested system is given in Fig.6.1. Note that two frame stores are required since the picture arrives from the camera in serial form. Only two mappings have been indicated, but this could be extended to any number with little increase in the size of the system. Since the 'cycle' time of the recognizer is small (considerably less than a microsecond) its decisions can be used in determining if and how the window should next be moved.

In designing the discriminators themselves, one has several options available. The simplest approach is to use random access memories (RAMs) in precisely the manner used in the computer simulations, i.e., each n-tuple constitutes the address input word for a RAM (see Fig.4.3 in section 4.4) and thus for 1:1 coverage of a 256 bit pattern matrix using 8-tuples, one requires 32 x 256-bit RAMs per discriminator. If 1 k-bit RAMs were used instead, the two extra address bits could be used to select one of four alternative fonts, for instance. Clearly RAMs have the advantage that their contents may be changed readily, i.e., a recognizer implemented

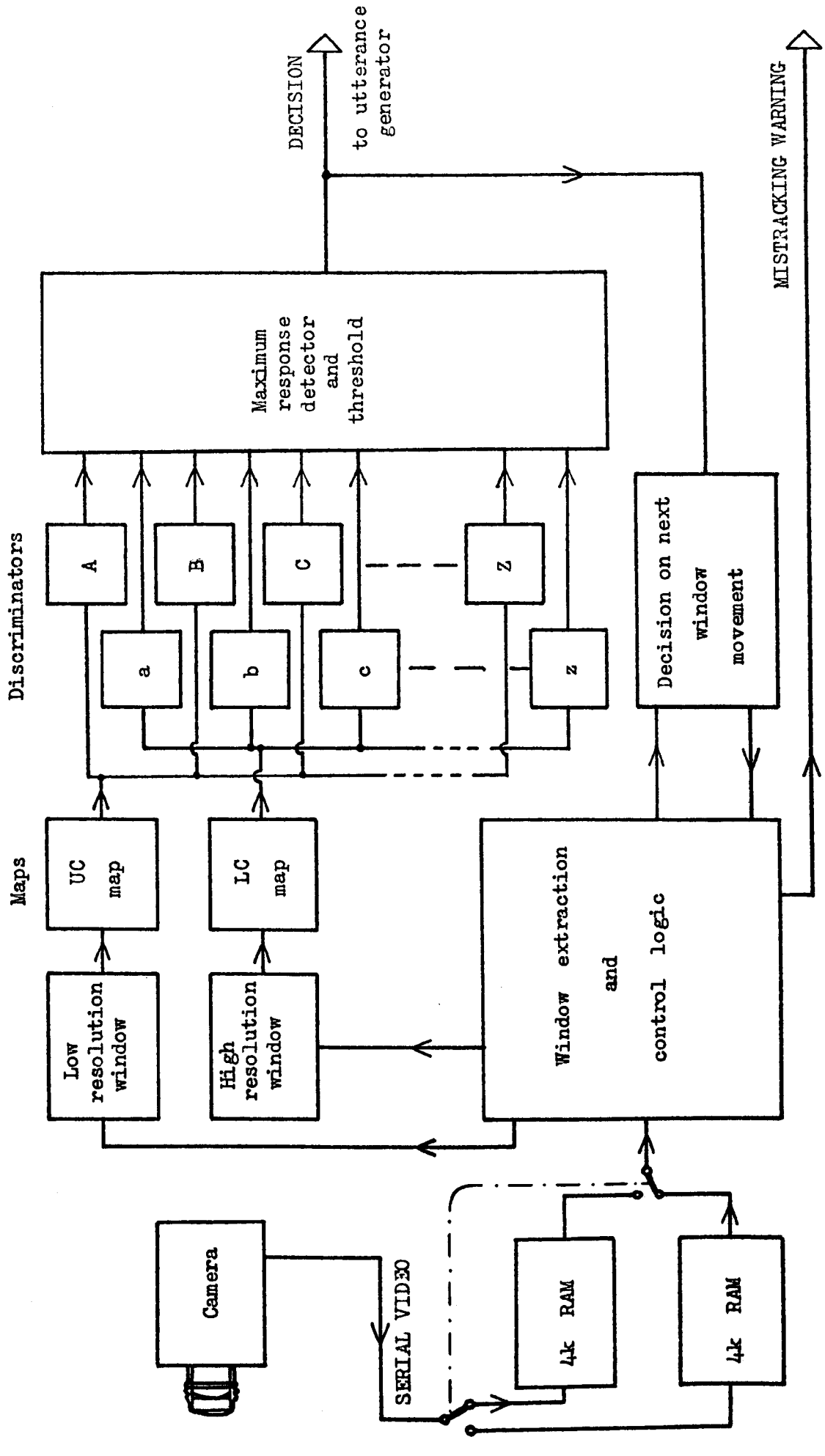


Fig. 6.1 Suggested Character Recognizer Configuration.

using RAMs would be trainable. This feature is useful in that it allows the user to add new type faces to the repertoire of the machine. However, it is apparent from the experiments reported in Chapter 5 that training the recognizer is no simple matter: one must be assured of the quality of the training patterns; one must have a means of setting class numbers, and finally one would not be able to use the optimisation techniques described in section 5.4, since they all rely on detailed analysis of either the training sets or of the contents of the trained recognizer. All of these tasks are best done on a computer simulation of the recognizer since this offers much more flexibility, and ready access to bulk storage for sets of patterns, etc. In practice, then, one would not desire to change the contents of the RAMs, in which case one could reduce the cost significantly by using read-only memories (ROMs), which would clearly not be trainable, but also would not suffer from the storage volatility problems associated with RAMs. All the training would take place on the computer simulation, and the final discriminator contents used as the basis for the manufacture (or programming) of the ROMs. Again the use of larger memory elements would facilitate the provision of several type faces.

There is a further important economy that can be made in the discriminator storage, provided that the number of states seen by each n-tuple during training is much less than the total number of possible states. If this condition is fulfilled one may use programmable logic arrays (PLAs) instead of ROMs. In a PLA one has a certain number of inputs (analogous to the address inputs of a ROM) and the possibility of decoding a certain number of states of this input word, this number being considerably less than the total number of possible states. Thus it can be equivalent to a ROM containing only the memory locations which have been set.

For example, reference to Table 5.12 in section 5.4 shows that in a typical experiment there were on average about 20 states seen by each n-tuple (the figures given under the heading 'Total States' are for all five discriminators added together), out of a total of 256 possible states. A ROM implementation would mean that the remaining 236 locations would contain zeroes and thus be wasted. In a PLA implementation one need only decode the n-tuple states seen during training, using only about one tenth of the storage otherwise needed, with a corresponding decrease in cost.

6.3 Future Work

Finally a few suggestions are made as to which areas of the research require more detailed study. Further work on the blind reading aid is required in deciding just how extensive its repertoire of type fonts should be, and how the decisions of the machine should be presented to the user (e.g., should decisions be reported - uttered - as soon as they are made, or should they be inhibited for a certain interval after each utterance, or even inhibited until there is a change between successive decisions?). Development of a suitable tracking algorithm is required in order to keep the input characters roughly centralised, with the generation of appropriate warning signals, as well as a means of adjusting the camera digitizing threshold automatically in order to accommodate variations in ambient light level and paper quality. These items require the development of the necessary hardware, as well as consideration of the appropriate human factors to assess their effectiveness.

A more detailed study of the various optimisation methods proposed in the current work is necessary, particularly on how groups of n-tuples act together in detecting pattern features, and what measurements should be made in order to select useful groups of

n-tuples. By this means mappings could be 'tailor-made' for a particular recognition problem with less reliance on the more intuitive methods used, for example, in selecting the combinations of random maps in section 5.4.3.

APPENDIX 'A'

Camera System Diagrams

A description of the solid-state camera system may be found in Section 4.2, and should be read in conjunction with the diagrams following.

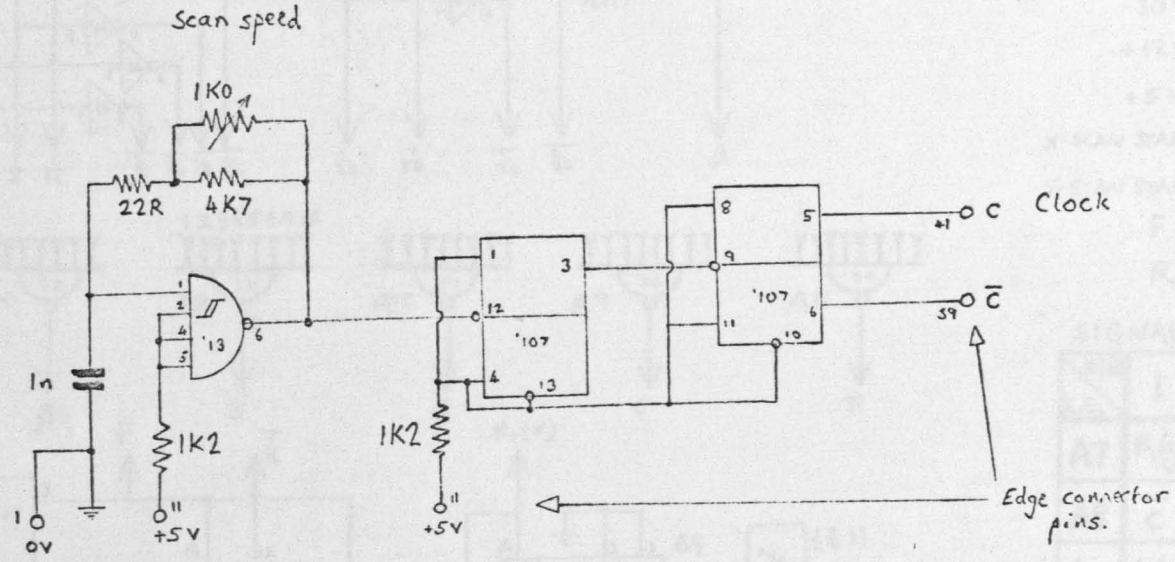
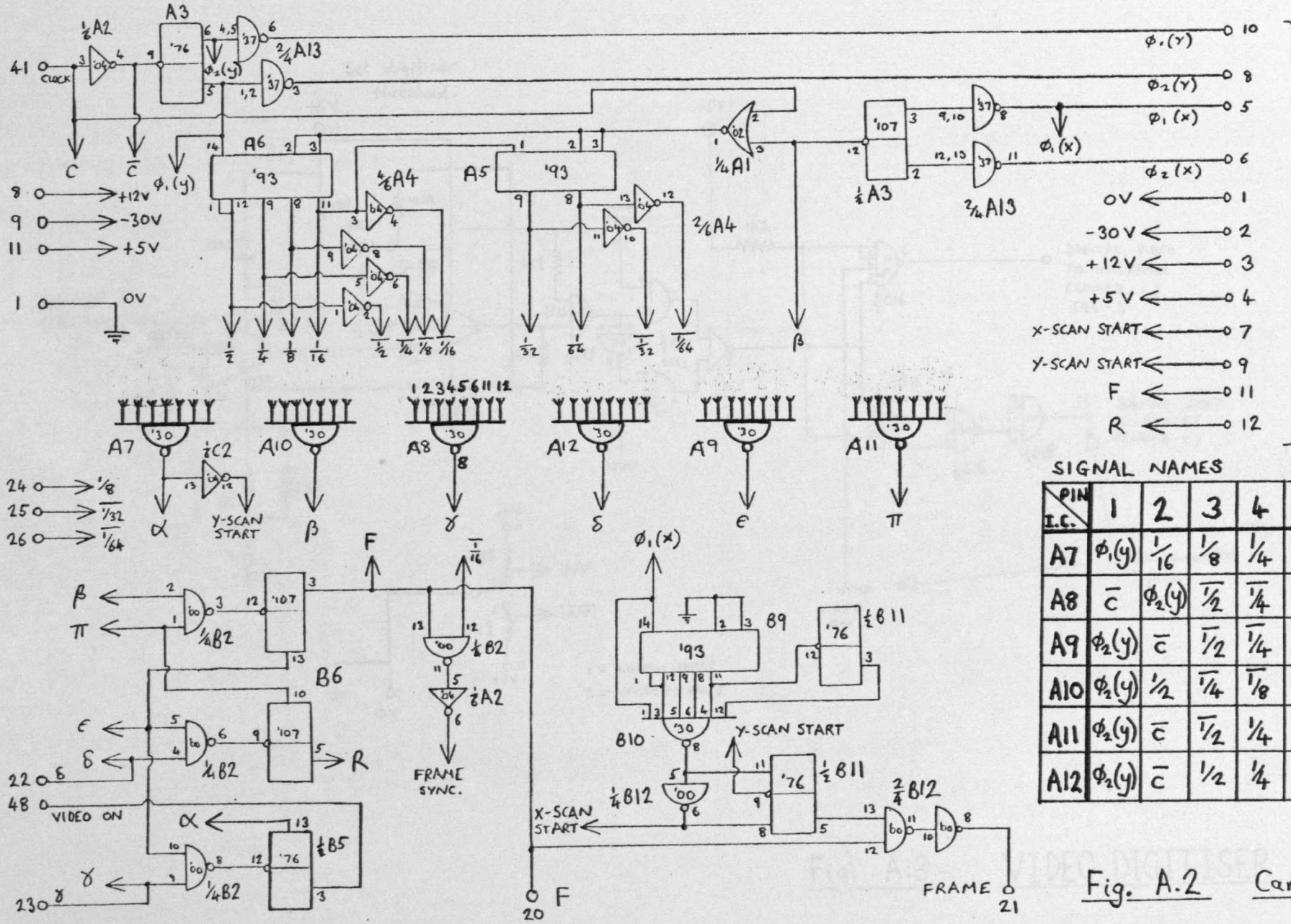


Fig. A.1 MASTER CLOCK (BOARD 2)



SIGNAL NAMES

PIN / I.C.	1	2	3	4	5	6	11	12
A7	$\phi_1(y)$	1/16	1/8	1/4	1/2	$\overline{1/64}$	$\overline{1/32}$	'1'
A8	$\bar{C}$	$\phi_2(y)$	$\overline{1/2}$	$\overline{1/4}$	$\overline{1/8}$	$\overline{1/16}$	1/32	$\overline{1/64}$
A9	$\phi_2(y)$	$\bar{C}$	$\overline{1/2}$	$\overline{1/4}$	$\overline{1/8}$	$\overline{1/16}$	1/32	1/64
A10	$\phi_2(y)$	1/2	$\overline{1/4}$	$\overline{1/8}$	$\overline{1/16}$	1/32	1/64	'1'
A11	$\phi_2(y)$	$\bar{C}$	$\overline{1/2}$	1/4	1/8	1/16	$\overline{1/32}$	$\overline{1/64}$
A12	$\phi_2(y)$	$\bar{C}$	1/2	1/4	1/8	1/16	$\overline{1/32}$	$\overline{1/64}$

Fig. A.2 Camera Driver.

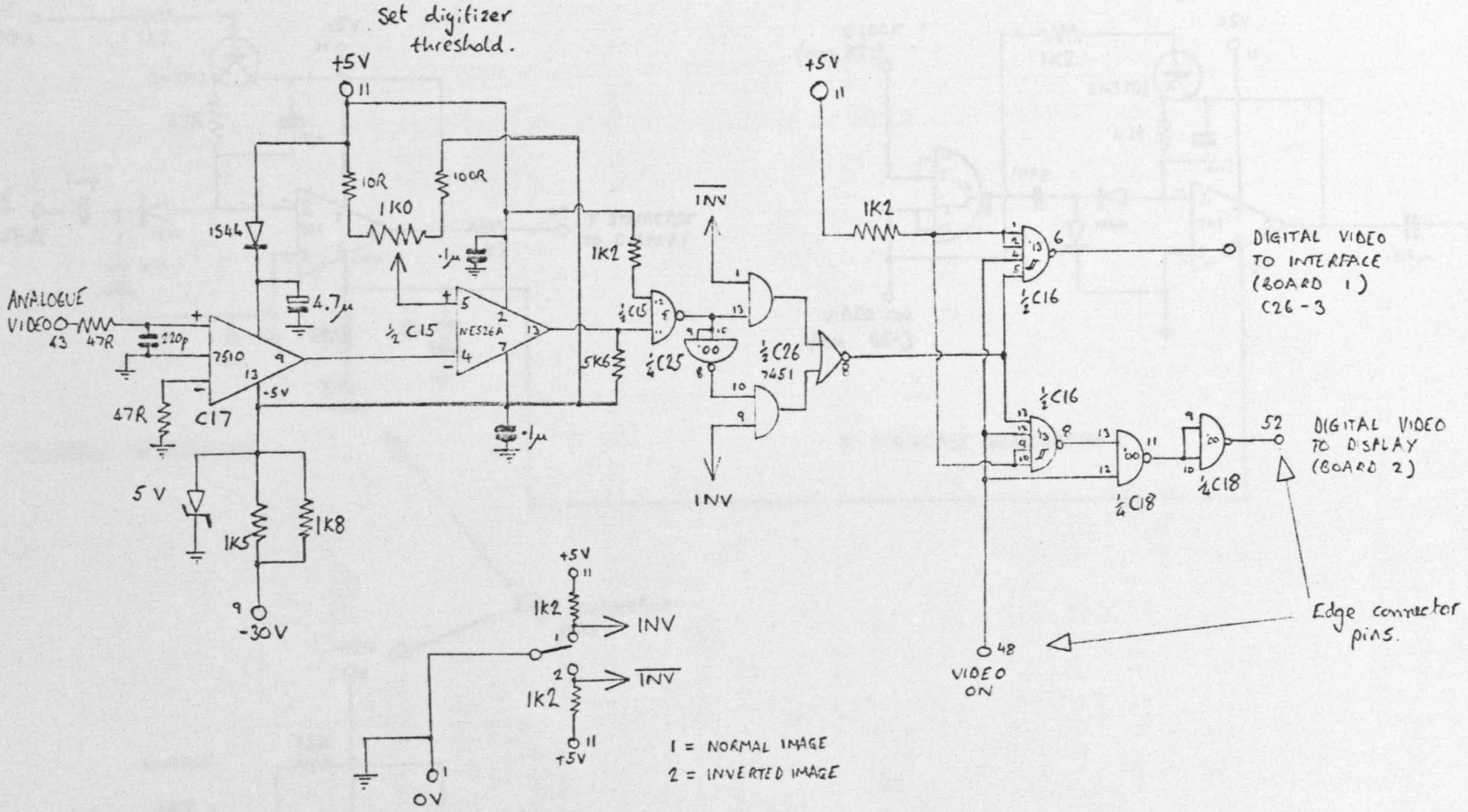


Fig. A.3 VIDEO DIGITISER (BOARD 1)

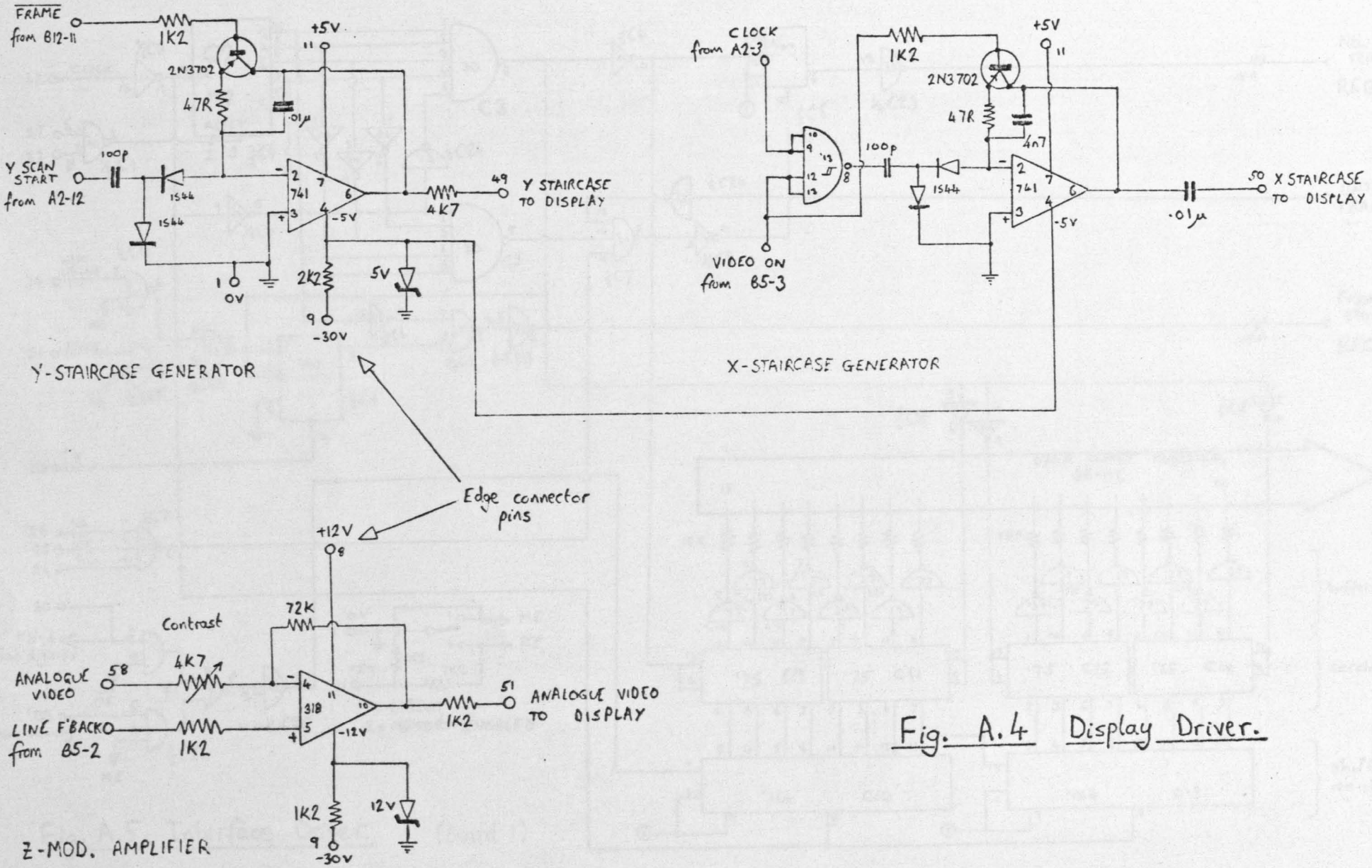


Fig. A.4 Display Driver.

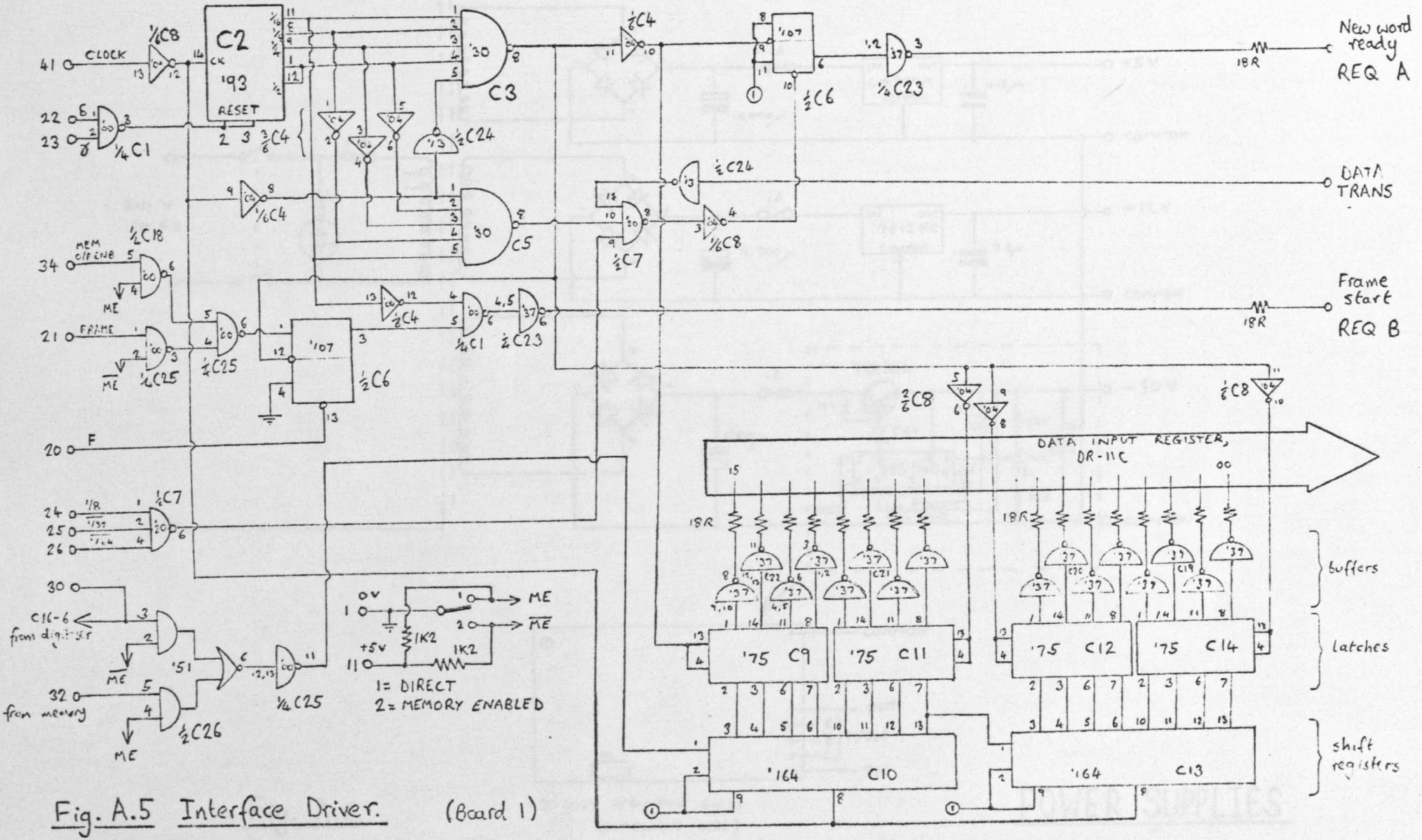


Fig. A.5 Interface Driver. (Board 1)

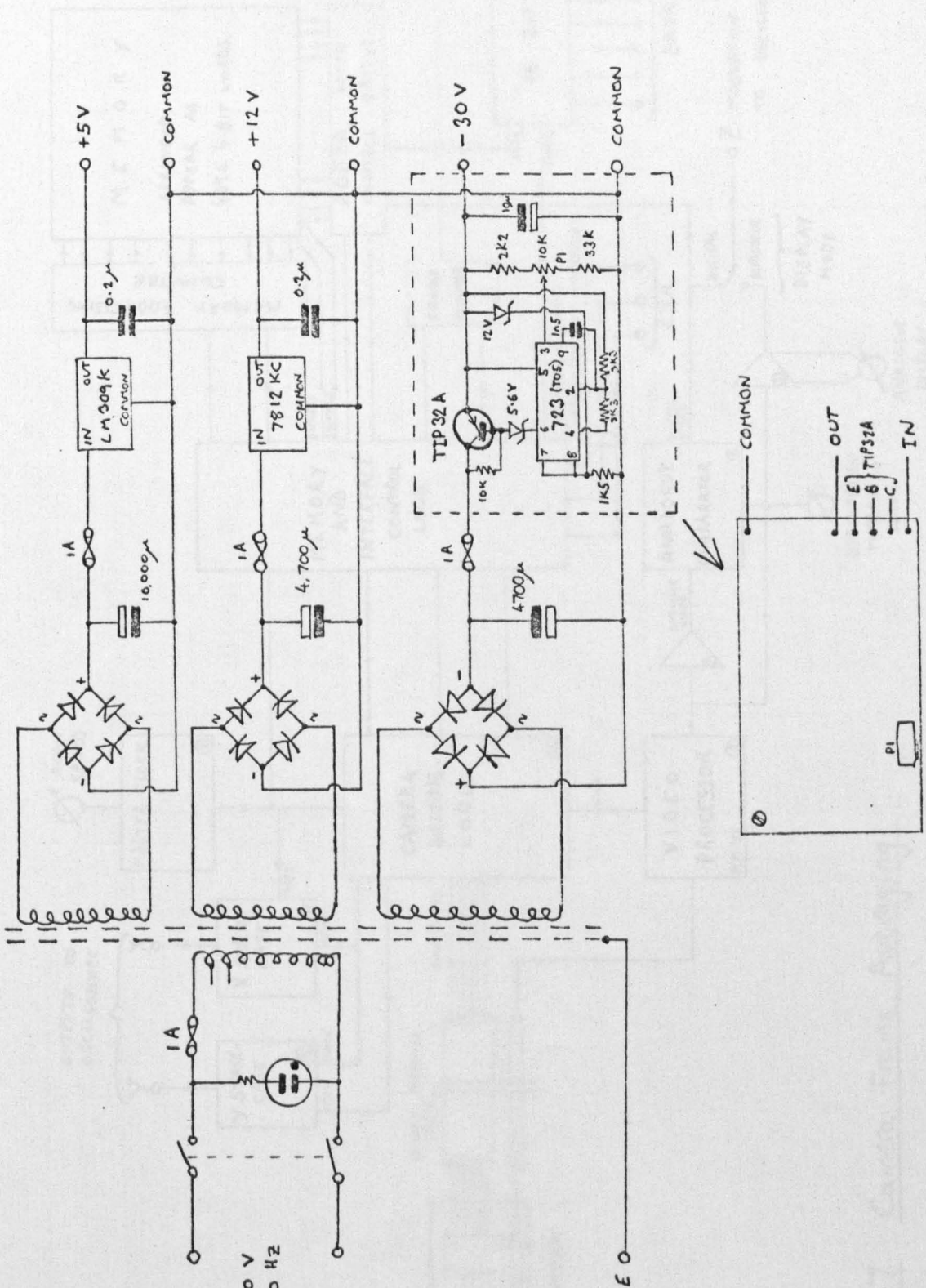


Fig. A.6

POWER SUPPLIES

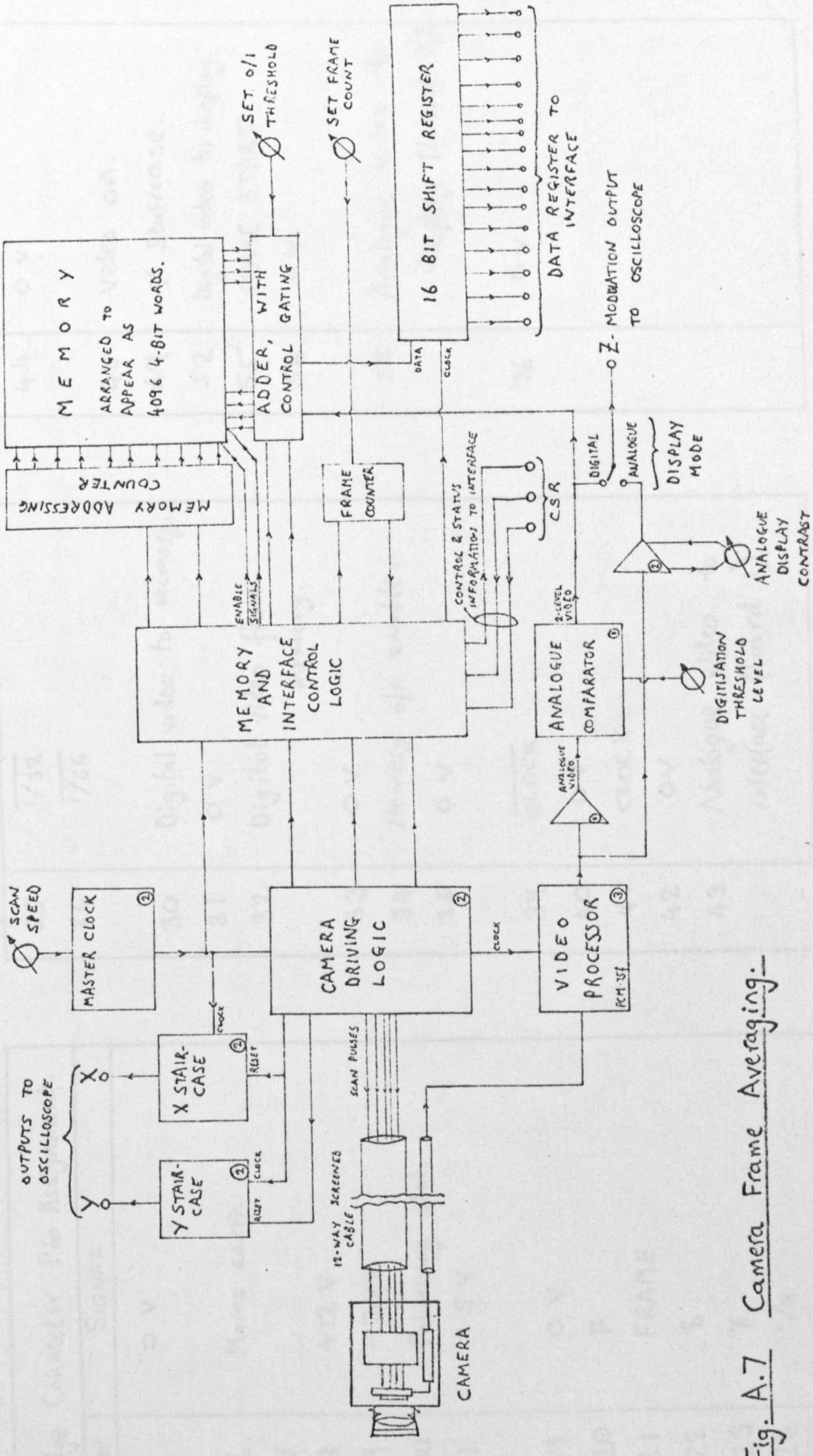


Fig. A.7 Camera Frame Averaging.

TABLE A.1

Edge Connector Pin Assignment.	
PIN	SIGNAL
1	0 V
6	Mains earth.
7	
8	+12 V
9	−30V
10	Polarising slot.
11	+5 V
19	0 V
20	F
21	FRAME
22	δ
23	γ
24	$\overline{1/8}$
25	$\overline{1/32}$
26	$\overline{1/64}$
30	Digital video to memory.
31	0 V
32	Digital video from memory.
33	0 V
34	Memory o/p enable.
35	0 V
39	$\overline{\text{CLOCK}}$
40	0 V
41	CLOCK
42	0V
43	Analogue video to interface (board 1)
44	0 V
48	Video on.
49	Y Staircase.
52	Digital video to display.
55	FRAME START
56	0 V.
58	Analogue video to display (board 2)
76	0 V

TABLE A.2

PCM-75f Analogue Video Processor. Edge Connector Pin Assignment.		
Pin	Colour	Signal
1	RED	+ 12 V
3	BROWN	− 30 V
4	BLACK	0 V
6	YELLOW	VIDEO IN
17	PURPLE	VIDEO OUT
23	ORANGE	CLOCK (= TTL TRIG.)

TABLE A.3

2D1 Camera Head. 'D-Type' Connector Pin Assignment.		
Pin	Colour	Signal
1	BLACK	0 V
2	BROWN	−30 V
3	RED	+12 V
4	GREY	+ 5 V
5	LIGHT GREEN	ϕ_1 (x)
6	DARK GREEN	ϕ_2 (x)
7	PINK	X SCAN START
8	WHITE	ϕ_2 (y)
9	BLUE	Y SCAN START
10	YELLOW	ϕ_1 (y)
11	PURPLE	F
12	ORANGE	R

TABLE A.4
INTERFACE CONNECTIONS

		25-way D-type connector		Interface edge connector		McMurdo connector		Berg connector	
INTERNAL NAME	DEC NAME	PIN NO.	COLOUR	Conn. no.	Pin no.	No.	Pin no.	No.	Pin no.
DATA TRANSMITTED	DATA TRANS.	1	ORANGE	2	23	2	D8	2	C
NEW DATA READY	REQ. A	2	BROWN	1	22	1	D7	1	LL
FRAME START	REQ. B	3	PURPLE	2	20	2	D9	2	S
		4							
BIT 0	IN00	5	YELLOW	2	3	2	A1	2	TT
BIT 1	IN01	6	BLUE	2	4	2	A2	2	LL
BIT 3	IN03	7	ORANGE	2	6	2	A4	2	BB
BIT 2	IN02	8	BROWN	2	5	2	A3	2	H
BIT 4	IN04	9	PURPLE	2	7	2	A5	2	KK
BIT 5	IN05	10	YELLOW	2	8	2	A6	2	HH
BIT 7	IN07	11	BLUE	2	10	2	A8	2	CC
BIT 6	IN06	12	ORANGE	2	9	2	A7	2	EE
BIT 8	IN08	13	BROWN	2	11	2	A9	2	Z
BIT 9	IN09	14	PURPLE	2	12	2	A10	2	Y
BIT 11	IN11	15	YELLOW	2	14	2	A12	2	V
BIT 10	IN10	16	BLUE	2	13	2	A11	2	W
BIT 12	IN12	17	ORANGE	2	15	2	A13	2	U
BIT 13	IN13	18	BROWN	2	16	2	D12	2	P
BIT 15	IN15	19	PURPLE	2	18	2	D10	2	M
BIT 14	IN14	20	YELLOW	2	17	2	D11	2	N
0 V	0 V	22	GREY	1, 2	2			2	UU, SS

APPENDIX B

A User's Guide to the JANSYS Programs.

Contents

B.1 JANSYS

JANSYS is a collection of utility routines running under the RT-11 operating system, providing support for the various applications programs (MULTI, EDIPIC, MULTI2, HAMAND, MAPPIT, FOSOC). Embedded within JANSYS is a flexible command structure which permits single operations to be carried out one by one, under complete conversational control. Alternatively, by making use of the looping and storage facilities, complex 'standard' programs may be built up and preserved as a disk file, for instance.

B.1.1 Description of the command structure.

The command structure was developed to allow conversational use of programs, such as the pattern classifier, which rely on a fairly large number of elementary operations. At the simplest level, a single command will initiate a single operation, such as reading in the next pattern. A command such as this is issued simply by typing the single letter mnemonic representing the command, followed by CR (carriage-return). When execution of the command is complete, a new prompting asterisk is issued by the system (assuming the absence of errors), e.g.,

```
*G
*
```

More than one command may be placed on the same line, merely by concatenating the appropriate symbols:

```
*GPYQGPY
*
```

Repeated execution of a single command may be obtained by typing the required number of iterations (maximum is 32767.) before the command symbol:

```
*30GPY2A
*
```

In the above example, the G command is executed 30 times, P and Y once, and A twice.

Iteration of a group of commands is achieved by enclosing it within brackets, thus:

```
*30[GPY]A
*
```

The group GPY is executed 30 times, and the single command A once.

Up to five nested levels of brackets are permitted, independently of single command iterations:

```
*10[X5[3Q25[GPY2[AQ5[XC10000F]]]RT5Y]]
*
```

The F command is executed 125,000,000 times.

To improve legibility, commas may be inserted within the string, and are treated as no-ops. A space within the string is treated as a command (and may thus be iterated) causing execution to halt for about 0.25 seconds. An "@" symbol occurring in a command string causes the console teletype to emit a bleep. All symbols occurring after a semicolon are ignored, so that comments may be inserted within a command line:

```
*25A,25[Gpy,3A,10[xQ ]],,,,,,,,,,,,;This is a comment and
*;                                  ;is ignored.
*
```

Upper case and lower case letters used as command symbols are totally equivalent.

Text placed between double quotes is printed exactly as typed, followed by a new line. Using a single quote instead of the terminating double quote suppresses the new line:

```
*AQ,2["Message"AX3Y],"This is all on on'FY3[QEW],"e line"
Message
Message
This is all on one line
*
```

B.1.2 Data Input.

Commands requiring data fall into two categories: those which pass the data on to the program for later use, and those which the data modify. A command to read a new pattern, for example, is of the first type, whereas a command to change the current class number, for example, is of the second type, because the new class number must be specified.

Commands of the first type normally obtain their data from one of the input/output channels. A particular command is permanently associated with a particular channel, but the device and/or file to which the channel is assigned may be changed at will (see 'Channel Assignments' below).

Commands of the second type always obtain their data from the same source as the command symbol itself, and the following method is used for specifying the data:

Assume that the symbol Z represents the command for changing the current class number. This function requires a single integer - the new class number - as data. If an integer immediately follows the Z, this is taken to be the datum:

```
*Z5,,,,,,,,,,,,,,,,,,,,,,,,,,;Sets class no.  5
*
```

If the Z is not followed by an integer, a new line is read from the same source as the command line, and the required integer looked for at the beginning of this line. Iteration of such a command would require one new line per iteration:

```
*4["Next class ',Z,3[GPY]],,,,;Class no. is 10,6,9,0
Next class Z10
Next class Z6
Next class Z9
Next class Z0
*
```

Note that if the command source is the console, as in the above example, the command symbol is printed as a prompt.

It is possible for the system to obtain its commands from a file, e.g., on disk, as described later. However, if in the above example the command string were on file,

i.e.,

```
4[*Next class ',Z,3[GPY]],,,,,,,,;Class no. is 10,6,9,0
10
6
9
0
```

the console output would be:

```
Next class Next class Next class Next class
```

Note the absence of prompt symbols, since nothing is now being read from the console keyboard, although the printing still occurs.

Console input of data may be requested explicitly, even if the command source is a file, by placing a question mark after the command symbol. For the above example, the command string on file is:

```
*4[*Next class ',Z?,3[GPY]],,,,,,;Get data from console
                                  ;TT.
```

and the console dialogue is:

```
Next class Z10
Next class Z6
Next class Z9
Next class Z0
```

Note that the command symbol is again used as a prompt. In summary:

Command string.	Command source.	Effect.
Z3	---	Use explicit data. See note below.
Z	Console	Issue 'Z' and read data from console.
Z	File	Read next line of file to obtain data.
Z?	---	Issue 'Z' and read data from console.

Note: constructions of the form 5Z3 should be avoided. The alternative, 5[Z3], will always work.

B.1.3 Channel Assignment.

JANSYS makes available three output channels, numbered 0,1,2, and three input channels, numbered 3,4,5. The function of each channel is defined within the program using it, e.g., channel 0 is used in EDIFIC for output of 64x64 bit patterns, whereas in MULTI it is used for output of discriminator response information. There is one channel, number 3, which is always opened for one purpose - the input of command strings from a file.

Each channel may be assigned independently to one device/file combination according to the normal RT-11 convention. In order to make a new assignment, the command symbol '!' must be executed. A new line is then read from the current command source (channel 3). This line must conform to the normal RT-11 convention for making channel assignments, e.g.,

```
*!
#,FILE3<,DT2:PAT57S
*
```

assigns FILE3 on the system disk to channel 1 (output), and PAT57S on Dectape 2 to channel 4 (input). A default extension is valid for each channel as specified within the program in use. If channel 1 was open prior to this assignment, it would be closed automatically before re-assignment. Channels 0,2,3,5 are not affected. Note that the above example used the console as the command source, hence the # prompt symbol.

Assigning a file to channel 3 causes the next command line to be read from that file. Any commands following the '!' in the current line are executed first. Note that channel 3 is initially open and assigned to TT: - the console. This assignment is restored automatically under any error condition.

B.1.4 Variables.

Twenty integer variables, taking values in the range 0 - 32767, are available for general use as counters, etc., and are accessed by the $x construction where x is the index number of the variable required, e.g., $3

accesses the fourth variable. This construction may replace any use of an integer within a command string, causing the value of the variable to be used. Values are assigned to variables thus:

```
*$0=3
*$$0=6
*
```

This places 3 in variable no. 0 and 6 in variable no. 3. Recursive dereferencing such as this is not recommended beyond about 5 levels, i.e., $$$$$0. The value assigned may be obtained as data (see 'Data Input' above), thus:

```
*"Value ',$4=?,,,,,,,,,,,;value obtained from console.
Value =12345
*

$6=,,,,,,,,,,,,,,,,,,,,,,,;value obtained from current
                           ;command source.
12345
```

Note that '=' is used as a demand symbol in the first example above.

The construction $x: causes the value of variable no. x to be printed on the console, followed by a new line (which is suppressed if the next character in the command string is a double quote). $x+ and $x- increment and decrement respectively variable no. x. The following example causes the value of each variable to be printed, using $0 as the index:

```
*$0=0,20[Variable no. '$0:" = '$$0:,$0+]
Variable no. 0 = 0
Variable no. 1 = 3751
Variable no. 2 = 7
 ....etc.
```

Using a variable to supply data is very simple, e.g.,

```
*$0=0,32[d$0,$0+]
```

The following string functions identically to the above:

```
*$0=0,32[d$0+]
```

This contracted form is, however, available only for the incrementing operation, which always occurs after the value of the variable has been used.

When using a variable as an iteration count, a bracketed group of commands will always be executed at least once, e.g.,

```
*$0=0,$0Q
```

causes the Q command to be executed zero times, whereas

```
*$0=0,$0[Q]
```

causes one execution of the bracketed group.

B.1.5 Error Handling.

When an error condition is detected, a descriptive message is printed on the console, and channel 3 is reassigned to the console (TT:) for the next command. In the case of channel assignment errors, e.g., ?FIL NOT FND?, a new * prompt symbol is issued, and the assignment string may be re-typed. In all other cases, a dot (.) prompt symbol is issued. There are only two correct responses to this: RE (restart) causes the program to continue with all data areas and channels (except no. 3) still valid. The response KI (kill) causes all i/o channels to be closed, and control to be returned to the RT-11 Monitor. The Monitor REENTER command may be typed, whereupon the terminated program will resume with all data areas valid, but with all channels except no. 3 closed.

If CONTROL/C is typed while a JANSYS program is running, an error condition is simulated, with the message 'BREAK IN'. Further action is as described above.

B.1.6 Miscellaneous Functions.

The following commands control various aspects of the system operation:

CONTROL/C — This causes the 'BREAK IN' message, followed by the prompt symbol '.', to which the only possible responses are 'RE' and 'KI'. A more detailed description is given under 'Error Handling' above.

CONTROL/E	This causes all text output on the graphics scroller to be echoed on the console printer. This includes commands typed in by the user. The function is disabled by pressing CTRL/E a second time.
CONTROL/S	When typed during the output of text, halts all operations until a second CTRL/S is typed. This allows the text to be inspected at the user's leisure.
ALTMODE or ESCAPE	Selects the program's 'immediate' mode. In this mode, typing a single character on the console effectively sends that character followed by a carriage-return. Thus simple commands may be single-stepped manually at a rapid rate. This mode is disabled at the next ALTMODE or ESCAPE, under any error condition, or whenever data are requested from the console (this includes execution of the ! command for file assignments).
RUBOUT	Deletes the previously-typed character (if any).
CONTROL/U	Deletes the whole of the line currently being typed. A ^U is printed, followed by a new line. The corrected line may then be re-typed from the beginning.
CONTROL/Q	Causes the 64x64 bit pattern to be displayed on the graphics screen. It may be removed by a second occurrence of CTRL/Q.

B.2 EDIPIC Version 04

This program contains the modules required for preparing files of patterns for later use as training or testing sets. It is capable of receiving and displaying 64x64 bit pictures from the solid state camera, extracting a movable 'window' of variable dimensions from this picture, and performing various other functions such as the averaging of several frames in order to reduce the effects of random noise. The commands available are described in detail below :

B.2.1 !

This allows channel assignments to be performed, as described in 1.3 above. The table below summarises the function of each channel, and the file extension which is assumed by default if none is specified explicitly.

Channel no.	Default extension	Function
0	C64	Output of 64x64 patterns.
1	C16	Output of 16x16 patterns.
2	---	Output of random maps and ASCII patterns.
3	CED	Input of commands.
4	C64	Input of 64x64 patterns.
5	C16	Input of 16x16 patterns.

The extension normally given to random map files intended for the recognizer programs RECIPE and RECIP4 is 'RMP'. 'MMP' map files intended for MULTI and MULTI2 must be generated using the program MAPPIT. The structures of 'C16', 'C64', and 'MMP' files are described in Section B.10.

B.2.2 X,Y

The execution of an X or Y command requires the input of a single integer in the range 1 to 16. This then becomes the new X or Y dimension of the 'window' being extracted from the current 64x64 bit pattern. Patterns being read from a C16 file are also truncated to the current window dimensions before being displayed. Note that execution of X or Y always causes the current window to be refreshed from the current 64x64 pattern, regardless of whether or not the previous window was obtained from a C16 file. The current class number is not affected.

B.2.3 G

This causes a new pattern to be read from channel 5 and displayed, subject to possible truncation (see B.2.2 above), at the top left of the VT11 screen. The current class number will be modified according to the contents of the byte reserved for this purpose within the C16 file structure (see Section B.10.1).

B.2.4 I

This causes one pattern to be read from channel 5, and discarded. The current pattern and class number are not affected. This command may save a considerable amount of time when large numbers of patterns have to be skipped.

B.2.5 Z

This requires the input of a single integer in the range 0 to 32, which becomes the new current class number. A subsequent O command (see B.2.10 below) causes this number to be placed in the special byte reserved within the C16 file structure for the class number. Note that a subsequent G command will modify the current class number.

B.2.6 H,J,K,L

In 'window' mode (see Section B.2.15), these commands cause the window currently being extracted from the 64x64 bit pattern to be shifted by one bit left (H), down (J), up (K), or right (L). In 'edit' mode, these movements are applied only to the small edit box.

B.2.7 F

This command causes a single 64x64 bit pattern to be read from channel 4, and displayed. The current class number is not affected, but the window is updated.

B.2.8 C

This obtains a single 64x64 bit pattern from the solid-state camera, and displays it. The current class number is not affected, but the window is updated. See B.1.6 above.

B.2.9 W

This outputs the current 64x64 bit pattern on channel 0, using the standard C64 file structure (see Section B.10.3).

B.2.10 O

This outputs the current window on channel 1, together with the current class number, using the standard C16 file structure (see Section B.10.1).

B.2.11 M

This requires the input of a single integer which constitutes the seed for a pseudo-random map, to be output on channel 2, using the standard RMP file structure. The 'RMP' map so produced is not suitable for use with the recognizer programs MULTI and MULTI2. 'MMP' maps for these must be produced by the program MAPFIT.

Note that a map requires both the starting seed and the current window dimensions in order to be fully specified. For example, the map produced by seed=4 for a 16x16 matrix would be referred to as map 4, 16x16.

B.2.12 D,P

These send the current 64x64 and 16x16 bit patterns respectively to channel 2, as legible ASCII files.

B.2.13 N,T

These are used in combination to receive and display averaged 64x64 bit patterns from the camera. The N command requires a single integer in the range 1 to 16, causing this number of frames to be read from the camera, and stored separately, without affecting the current display. The T command again requires a single integer in the range 1 to 16, which becomes the threshold. The stored frames are now inspected, and the averaged pattern displayed. A particular bit of the averaged pattern is set only if the corresponding bit of the input frames was set a number of times not less than the threshold value.

B.2.14 Q

This complex command was implemented to allow the camera to be adjusted, e.g., pointed at a new class of pattern, whilst a possibly large command loop is being executed. It implicitly performs N and T commands repeatedly until a CONTROL/Y is typed at the console teletype. The frame count value (data for the N command) must be in variable 8 ($8, see 1.4 above), and the threshold value (data for T) must be in variable 9 ($9) before execution of Q. For example, the following sequence of commands creates a file containing 100 patterns of each of classes 1 to 4, using 12 and 6 as the averaging parameters, and allowing the camera to be repositioned between classes.

```
*!
#,PATOUT<
*$8=12,$9=6
*$0=1
*4[z$0,"Get class no. "$0:,$0+,q,100[N12,T6,0]]
Get class no. 1
Get class no. 2
Get class no. 3
Get class no. 4
*
```

B.2.15 A,/,\,R,S

Alternate use of the A command causes the program to switch between 'window' mode, which is the initial mode, and 'edit' mode, which may be identified by the presence of a small flashing box on the 16x16 display, enclosing exactly one pattern element. In edit mode, the move commands H,J,K,L described in Section B.2.6 do not move the entire window, but just the small edit box. The / command sets the enclosed pattern element, whilst the \ command clears it. After each of these commands, the box moves by one element in the same direction as the previous move (however long ago this may have been). / and \ are permissible only in edit mode. The S command resets the box to the bottom left hand corner of the window, whilst the R command also clears the whole window. These commands are permissible at any time.

B.3 MULTI Version 01

This is the pattern classification program. The characteristics of the classifier required for a particular application may be specified during an optional dialogue phase when the program is initially loaded. The characteristics under control in this way cannot be changed at a later stage without reloading the program. They are:

i) Restoration of old RAM file. This restores the discriminator contents and map exactly as saved by the 'O' command (see Section B.3.15) during a previous run. Questions ii), iii), iv) and v) are skipped if an old RAM file is restored in this way.

ii) Window X and Y dimensions, each in the range 1 to 16. Default: 16x16.

iii) Classifier n-tuple size, in the range 4 to 8. Default: 8.

iv) Number of RAMs per discriminator. Typing 'CR' causes the automatic setting of a 1:1 connection between the pattern elements and the RAM inputs.

v) Number of discriminators in use. The number available is dependent upon the amount of free memory available, which in turn is dependent upon ii), iii) and iv) above, but a minimum of 10 is guaranteed. The absolute maximum number of discriminators is 32. Default: 10.

vi) Renaming of discriminators. By default, the classes are referred to by number, i.e., 1,2,3,4,...,30,31,32. Renaming them alphabetically causes them to be referred to as A,B,C,D,.....,X,Y,Z,27,28,...,31,32. The classes may, however, be renamed in any way the user desires, using two characters per name, even to the extent of giving several different classes the same name.
Note that class 0 is permanently named '?' since it is a special class having no discriminator, and normally associated only with a 'don't know' condition (i.e., for a test pattern of unknown class, or to indicate an ambiguous response to a pattern). Since it is not possible to alter the contents of any discriminator, either by teaching or clearing, whilst the current class is 0, this constitutes a means of 'write-protecting' the

storage against inadvertant corruption.

vi) Absolute threshold. Replying with 'Y' to this question causes the program to use an absolute threshold when making all classification decisions (X,Y,F,A and E commands below). Any other response causes a relative threshold to be used. In both cases the value of the threshold is held in variable 10 ($10), whose value is set initially to zero. Default: relative threshold.

The commands available are as follows:

B.3.2 !

This allows channel assignments to be performed, as described in B.1.3 above. The table below summarises the function of each channel, and the file extension which is assumed by default if none is specified explicitly.

Channel no.	Default extension	Function
0	RES	Output of discriminator responses.
1	RAM	Output of discriminator contents.
2	LOG	Output of log file.
3	CRE	Input of commands.
4	C16	Input of 16x16 patterns.
5	MMP	Input of mapping information.

The structures of RES, C16, and MMP files are described in Section B.10. The effect of opening channel 2, the log, is to send all text appearing on the VT11 scroller simultaneously to the specified file.

B.3.3 G

This causes a new pattern to be read from channel 4 and displayed, subject to possible truncation (dependent upon the X and Y dimensions set up during the initial

dialogue phase - see B.3.1 above), at the top left of the VT11 screen. The current class number will be modified according to the contents of the byte reserved for this purpose within the C16 file structure (see Section B.10.1). If a pattern is encountered whose class number does not lie within the range 0 to the number of classes set up during the initial dialogue, the current class number is set to 0. An error condition (?CLASS NO.?) arises should teaching be attempted for this class (see B.3.8 below).

B.3.4 I

This causes one pattern to be read from channel 4, and discarded, without affecting the current pattern or class number. This command may save a considerable amount of time when large numbers of patterns have to be skipped.

B.3.5 Z

This requires the input of a single integer which becomes the new current class number. It must lie in the range 0 to the number of classes set up during the initial dialogue (see B.3.1 above). The data required by this command is always an integer, regardless of any renaming of the discriminators. Note that a subsequent G command will modify the current class number.

B.3.6 H,J,K,L

These commands cause the window currently being extracted from the 64x64 bit pattern to be shifted by one bit left (H), down (J), up (K), or right (L).

B.3.7 C

This obtains a single 64x64 bit pattern from the solid-state camera, and displays it. The current class number is not affected, but the window is updated. Note that the 64x64 display may be switched on or off by use of the CONTROL/Q function (see B.1.6 above). This is often desirable since the generation of the display file is very time-consuming.

B.3.8 T

This command teaches the discriminator whose class number is currently selected, using the current pattern as displayed at the top left of the VT11 screen. Attempting to teach class 0 (no discriminator) generates the ?CLASS NUMBER? error message.

B.3.9 B

This command performs exactly the same functions as a G command (B.3.3) followed by a T command (B.3.8). It speeds up the teaching, particularly when large numbers of patterns have to be processed, since it avoids the use of bracketed command groups (see B.1.1 above), i.e.,

```
*10000[GT]
```

is more time-consuming than

```
*10000B
```

B.3.10 Q

This clears the entire discriminator storage, unless class 0 is currently selected, in which case an error condition arises.

B.3.11 M

This reads one map from channel 5, using the standard MMP file structure (see Section B.10.4). This mapping is then invoked whenever a new pattern is generated (by G,B,V,H,J,K,L, or C commands), in transforming the pattern into the input vector for each discriminator. Note that in this version of the pattern classifier the same mapping is used for all discriminators. It is important that the map file is correct for the matrix dimensions currently set. The ?BAD MAP? message does not necessarily appear if this is not so. See Section B.2.11.

B.3.12 D

This sends to the display, as a series of six-digit octal numbers, the contents of one RAM within the discriminator selected by the current class number. The RAM is specified by an integer which is given as data for the D command. The following sequence dumps the contents of RAM 0 of the class 6 discriminator:

```
*Z6"Class 6, RAM 0"D0
Class 6, RAM 0
000001  007410  150014  006177
*
```

The number of words dumped does of course depend upon the n-tuple size (see B.3.1 above), whilst the number of RAMs in one discriminator depends upon both the n-tuple size and the window dimensions. No checks are made on the validity of the data supplied for the D command.

B.3.13 X,Y,P,A,E,S,V

These seven commands are concerned with the handling of the classifier's response to the current pattern. All except Y cause a discriminator output vector (one bit per RAM) to be generated for each class, using the current input vector, which is set up whenever a new pattern is generated (see B.3.3 and B.3.11 above).

B.3.13.1 X

This updates the display, immediately beneath that of the current pattern, of the name of the discriminator having the greatest response to the current pattern. The response is evaluated by adding up the number of bits set in the discriminator output vector. The name '?' indicates that two or more discriminators have equal responses.

B.3.13.2 Y

This merely prints out on the scroller (and on the log file, if open) the current class number and the maximum response exactly as displayed beneath the current pattern. It does not cause the response to be updated (see B.3.13.1 above).

B.3.13.3 P

This performs an implicit X command (B.3.13.1), then prints on the scroller (and on the log file, if open) the numerical response of each discriminator, for example, in a six-class system:

```
*GP
  15   7  13  16   7  10
*Y
CLASS     C
MAX RESP. D
*
```

B.3.13.4 A

This again performs an implicit X command (B.3.13.1), then prints the output vector of each discriminator, one per line, as a sequence of 0s and

1s corresponding to the outputs of the individual RAMs. The total number of bits set in each output vector is printed at the end of the line. The above example would become:

```
*GA
1111111111011111 15
1100100100001110 7
1011101111111101 13
1111111111111111 16
0101100010001110 7
1100001111010111 10
*
```

B.3.13.5 E

This command is intended to enable the overall recognition ability of a particular configuration to be established quickly, without having to collect and compare individual responses to test patterns. It performs an implicit X command (B.3.13.1), then compares the name of the current class with the name of the discriminator having the maximum response after the application of the appropriate threshold. If they are the same, variable 9 ($9 - see B.1.4) is incremented. If two or more discriminators have the same maximum response, variable 8 ($8) is incremented instead, regardless of whether any of the responses is correct. For example:

```
*$8=0$9=0
*2[GXYE$8:" '$9:,""]
CLASS     C
MAX RESP. C
    0     1

CLASS     C
MAX RESP. ?
    1     1

*
```

B.3.13.6 S

This evaluates the numerical response of each discriminator to the current pattern, and outputs these, along with the current class number, on channel 0, using the standard RES file structure (see Section B.10.2). On the first execution of an S command after the opening of channel 0 to a new file, a header containing the n-tuple size, window dimensions, date, class names, and a title is written on the file. This title, one line of text up to 68 characters long, is given as the next line of the command source (channel 3). If this is the console, a prompting message is printed. The current values of variables (see B.1.4) may be inserted in the title by using the '$' symbol: the first occurrence of '$' within the title is replaced by the value of variable 0, the second by the value of variable 1, and so on. The example program in Section B.11.2 shows the use of this facility.

B.3.13.7 V

This command performs exactly the same functions as a G command (B.3.3) followed by an S command (B.3.13.6). The advantages of doing this are as described for the B command (see B.3.9).

B.3.14 W

This command trains the current discriminator on the current pattern if the overall response to the pattern is incorrect (after application of the threshold). Variable 6 ($6) is incremented if training occurs.

B.3.15 O

This saves the map, plus the entire discriminator contents, on channel 1. A two-line title must be supplied. Such a file may be restored only via the initial dialogue.

B.3.16 R

This command performs implicit G and X commands, then inspects the response obtained. If it is correct or a rejection, the process is repeated. If the response is incorrect, the loop is terminated and the current class is changed to that of the discriminator giving the (incorrect) maximum response.

B.3.17 F

The F command executes C, X commands implicitly, and moves the window if necessary (using a J or K command) in an attempt to centralise the pattern and improve the responses. If a unique response is obtained, the window is not moved. Otherwise the top and bottom rows of the window are inspected and if the bottom row is not blank the window moves down one step, otherwise if the top row is not blank, it moves up one step. If the top and bottom are both blank, it does not move.

B.3.18 U

This teaches the current class discriminator to respond with zeroes to the current pattern, and is thus an 'unteach' function since it causes memory bits to be cleared rather than set.

B.3.19 N

This command obtains the response to the current pattern, and if it is incorrect (but not a rejection), the discriminator having the maximum response is untrained (U command) on the pattern.

B.4 MULTI2 Version 01

This program is the multiple-map version of MULTI, and announces itself as such when loaded. The operation of MULTI2 is identical to that of MULTI as described in Section B.3, apart from a few exceptions detailed here. The structure of MULTI2 allows the simultaneous existence of several (currently up to six) independent mappings between the current pattern and the input vector. This naturally incurs higher overheads in both execution time and storage than MULTI, and for this reason the number of discriminators available may be somewhat less. The initial dialogue differs slightly from that of MULTI in that currently a 1:1 mapping is always assumed, and there are two further questions:

i) Number of maps. The limit is currently six.

ii) Map/discriminator correspondence. For each discriminator in use, the user must type the number of the map (starting at 0) which is to be associated with that class, e.g., for a six-class system using three maps:

```
1?0
2?0
3?1
4?2
5?0
6?2
```

It should be noted that the map numbers used here are in no way related to the numbers used in creating the random maps (see Section B.6.3).

The following command differs from its counterpart in MULTI:

B.4.1 M

This single command reads all the required maps from channel 5, using the standard MMP file structure (see Section B.10.4). All the maps must therefore exist in the correct sequence on the appropriate single file. It is important that the map file is correct for the matrix dimensions currently set. The ?BAD MAP? message may occur if it is not.

B.5 HAMAND Version 02

This program contains the modules required for analysing sets of patterns used for training and testing purposes in MULTI or MULTI2. It is able to generate both averaged and accumulated patterns for each of the 32 possible classes, calculate Hamming distances between patterns, edit files of patterns on the basis of Hamming distance criteria, and generate histograms showing the Hamming distance distribution of patterns within a given set. HAMAND maintains for each class both an accumulated pattern, to which the current pattern may be added by the A command (see B.5.8 below), and, derived from this, an average pattern. The commands are as follows:

B.5.1 !

This allows channel assignments to be performed, as described in B.1.3 above. The table below summarises the function of each channel, and the file extension which is assumed by default if none is specified explicitly.

Channel no.	Default extension	Function
0	HFR	Output of binary accumulator contents.
1	C16	Output of 16x16 patterns.
2	PIC	Output of ASCII patterns and histograms.
3	CHA	Input of commands.
4	C16	Input of 16x16 patterns.
5	---	Not used.

The structure of C16 files is described in Section B.10.1.

B.5.2 G

This causes a new pattern to be read from channel 4 and displayed at the top left of the VT11 screen. This is known as the current pattern. The current class number is modified according to the contents of the byte reserved for this purpose within the C16 file structure (see Section B.10.1). See B.5.4 below for the effects of changing the current class number.

B.5.3 I

This causes one pattern to be read from channel 4, and discarded, without affecting the current pattern or class number. This may save a considerable amount of time when large numbers of patterns have to be skipped.

B.5.4 Z

This requires the input of a single integer which becomes the new current class number, and must lie in the range 0 to 32. Note that class 0 is named '?', and classes 1 to 26 are named 'A' to 'Z'. Whenever the current class changes, the average pattern, if any, for that class is displayed automatically at the top right of the VT11 screen. As in MULTI, class 0 is a dummy, and therefore has no average pattern, so the A command described below is not allowed whilst this class is selected.

B.5.5 O

This outputs the current pattern on channel 1, together with the current class number, using the standard C16 file structure (see Section B.10.1).

B.5.6 R

This resets the accumulated and average patterns for the current class.

B.5.7 C

This command resets the accumulated and average patterns for all classes, leaving the system in class 0.

B.5.8 A

This command adds the current pattern to the accumulated pattern of the current class, increments an internal counter for the number of patterns of this class seen, and regenerates the appropriate average pattern. A particular bit within the average pattern is set only if the corresponding bit in each of the patterns of the same class previously seen was set at least half the time. Note that the display of the average pattern is not updated - a V command (see B.5.9 below) is necessary to achieve this.

B.5.9 V

This displays the average pattern for the current class at the top right of the VT11 screen.

B.5.10 X

This displays the accumulated pattern for the current class at the top right of the VT11 screen. Each bit within the pattern takes on one of eight intensities according to the number of times the corresponding bit was set in each of the patterns of the same class previously seen.

B.5.11 Y

This outputs the average pattern of the current class (not class 0) in legible ASCII format on channel 2.

B.5.12 E

This command repeatedly reads patterns from channel 4 until the end of file is reached, then prints, on the scroller, the number of patterns of each of the 32 possible classes within the file. The current pattern is not affected, nor is the current class number.

B.5.13 H

This displays the Hamming distance (number of bits differing) between the current pattern and the average pattern of the current class.

B.5.14 W

This command enables files to be edited such that they contain patterns with particular Hamming distance characteristics. Two parameters are required, and these are held in variables (see B.1.4). The W command implicitly performs repeated G (B.5.2) and H (B.5.13) commands either until the end of file is reached (error condition) or a pattern is seen whose Hamming distance from the average pattern of its class falls between the low and high limits as held in variables 8 and 9 respectively ($8 and $9). The example program in Section B.11.3 shows the use of this facility.

B.5.15 Q,S,T

These commands are concerned with the generation of histograms showing the Hamming distance distributions of the patterns within particular sets. Q clears the histogram storage so that a new start can be made. S performs an implicit H command (B.5.13) then adds the Hamming distance value so obtained to the distribution histogram being built up. T outputs on channel 2 the current histogram, in which the y axis corresponds to the frequency of occurrence of patterns falling within a particular Hamming distance range, as marked on the x axis. The following example builds up the average patterns for each of 26 classes, using the file TSTPAT.C16, then reopens the file so that the distribution histogram may be constructed using the same patterns. The histogram is then sent to the line printer, together with a title, which is given as the next line of the command source (channel 3). A prompting message is printed if, as in this case, the command source is the console teletype.

```
*QC;                              ;Reset everything.
*!;                               ;Open the input file.
#,TSTPAT
*26[100[GA]];                     ;Build up the average
                                  ;patterns.
*!;                               ;Reopen i/p file and open
                                  ;o/p file.
#,,LP:<,TSTPAT
*2600[GS];                        ;Build up the histogram.
*T;                               ;and output it.
Type histogram title.
This is the title: one line of text.
```

B.5.16 F

This command outputs the current average pattern on channel 1, together with the appropriate class number, in the normal C16 file format (see Section B.10.1).

B.5.17 J

This outputs the current Hamming distance value, as obtained by an implicit H command, in text form on channel 2.

B.5.18 B

This outputs on channel 2 in text form the current accumulated pattern in which each pattern element is represented by a five digit decimal frequency value. The class number and number of patterns accumulated are also given.

B.5.19 D

The D command dumps the current accumulated pattern in binary form on channel 0. This is normally done for each class, and the file produced is used as input for the program ACCPRO, which calculates the across-class standard deviation of each pattern element.

B.6 MAPPIT Version 02

MAPPIT is the only program capable of producing the 'MMP' format maps (see Section B.10.4) suitable for the recognizer programs MULTI and MULTI2. The currently-held mapping is displayed, either in full or in part, at the top of the graphics display in a matrix of the same size as the patterns for which the map is intended. Each element in the matrix is the number of the RAM (or n-tuple --- these terms are effectively synonymous) to which the corresponding pattern element is connected. Thus for a 6-tuple system there will be six zeroes, six ones, and so on.

B.6.1 !

This allows channel assignments to be made in the usual way (see Section B.1.3).

Channel no.	Default extension	Function
0	MMP	Output of maps.
1	---	Output of text.
2	---	Not used.
3	CMA	Input of commands.
4	MMP	Input of maps.
5	---	Not used.

The structure of MMP files is described in Section B.10.4.

B.6.2 C

This command reconfigures the system by holding a short dialogue with the user. The parameters under control are: X dimension (1 to 16), Y dimension (1 to 16), n-tuple size (4 to 8), and number of RAMs per

discriminator. If the product of X and Y is IPOINT, the number of pattern elements, and R is the number of RAMs per discriminator, then the restriction on R is:

$$R \leqslant IPOINT/n$$

This does not, however, preclude the possibility of connecting one pattern element to more than one RAM input. Future versions of MAPPIT may permit a more flexible 1:many mapping structure.

B.6.3 M

This requires the input of a single integer which is the seed for a pseudo-random map which becomes the current map displayed on the screen. Note that no i/o action is taken.

B.6.4 R

This generates a regular mapping, and is otherwise the same as M (see Section B.6.3).

B.6.5 O

This outputs the complete current map on channel 0. Note that the complete map possesses two initial words containing the values of nR and IPOINT (see Section B.6.2).

B.6.6 I

This inputs a complete map from channel 4. Note that the system must be configured correctly for the map parameters nR and IPOINT (see Section B.6.2).

B.6.7 N

This requires the input of a single integer which becomes the selected or current n-tuple. The map display is amended to show only those pattern elements connected to the current n-tuple.

B.6.8 X

This restores the full map display, so that there is no current n-tuple.

B.6.9 E

This first requires the input of a single integer which is interpreted as a bit number within the current n-tuple. A second integer is then demanded, this being the new pattern element to which the specified bit of the current n-tuple is to be re-connected. The pattern elements are numbered from zero in the bottom left-hand corner, up the first column, then up the second column, and so on. The E command is not allowed when the full map is being displayed and there is thus no current n-tuple.

B.6.10 D

This requires the input of n integers, these being interpreted as the new pattern elements to which the current n-tuple is to be re-connected. See Section B.6.9 for pattern element numbering. The D command is not allowed when the full map is being displayed.

B.6.11 P

This outputs the current display of the complete or partial map exactly as it appears on the screen, in text form on channel 1.

B.6.12 H

The H command outputs on channel 0 just the map file header; i.e., nR and IPOINT (see Section B.6.2). This must be followed by sufficient S commands (see Section B.6.13) to build up a complete map piecemeal.

B.6.13 S

This outputs just the current n-tuple on channel 0. A map file built up piecemeal in this way must start with the appropriate map header (see Section B.6.12).

B.7 FOSOC Version 01

This program is effectively a version of MULTI in which the individual locations of the RAMs are 16-bit words instead of single bits, and the teach process increments (instead of setting) the location addressed within each RAM by the current pattern (via the usual 'MMP' mapping). In this way the occurrence frequencies of all the n-tuple states may be accumulated, although due to the increased storage requirements, only one class (discriminator) can be handled at a time. At the end of training on one class, the memory contents are dumped on the output file, the storage cleared, and the next class processed. The output file is then used as input for the program FRAN, which calculates statistics based on the whole file (see Section B.8). Alternatively, but less satisfactorily, FOSOC may be used directly to output the n-tuple state occurrence frequencies after training on each class, by use of the S command (see Section B.7.8). It is important that the system be configured correctly (C command - see Section B.7.2) before reading in the map or any patterns.

B.7.1 !

This performs the i/o channel assignments in the usual way (see Section B.1.3).

Channel no.	Default extension	Function
0	FRE	Output of frequency counts (binary).
1	LST	Output of text.
2	---	Not used.
3	CFO	Input of commands.
4	C16	Input of 16x16 patterns.
5	MMP	Input of maps.

B.7.2 C

This command should be used before any of those following. It configures the system by means of a short dialogue in which the parameters specified are: X and Y dimensions, n-tuple size, and number of RAMs (there is only one discriminator available at a time). Replying to the latter question with 'CR' automatically sets up a 1:1 configuration in which nR=IPOINT (see Section B.6.2). Re-configuration also clears the accumulator storage, for which reason the program asks if the user is sure that he wishes to re-configure.

B.7.3 Q

This clears the entire accumulator storage.

B.7.4 M

This reads in an MMP format map on channel 5, checking that the parameters nR and IPOINT correspond to the FOSOC system as currently configured (see Section B.6.2).

B.7.5 G

This reads in the next pattern from channel 4. Its class number is ignored.

B.7.6 I

This skips one pattern on channel 4.

B.7.7 A

This accumulates the current pattern, or rather the current states of all the n-tuples set up by the current pattern.

B.7.8 S

This produces a direct listing on channel 1 of the occurrence frequency of every n-tuple state seen since the store was last cleared, irrespective of pattern class. The following command string shows how a set of n-tuple state occurrence frequencies could be built up for each of 10 classes, given that there are 200 examples of each class on the input file, and that the system is already configured:

```
*!
#,LP:<,PATIN,R1214
*M
*10[Q,200[GA]S]
*
```

B.7.9 F

This outputs the accumulator contents in binary on channel 0 in a format suitable for reading by the analysis program FRAN (see Section B.8). This is the recommended means of obtaining the n-tuple state occurrence frequencies. The following example processes the same file as in Section B.7.8:

```
*!
#ACCOUT<,PATIN,R1214
*M
*10[Q,200[GA]F]
*
```

B.8 FRAN

This FORTRAN program does not operate under the JANSYS run-time package, but receives all of its commands through an initial dialogue with the user. It is used to process the n-tuple state occurrence frequency files (extension 'FRE') produced by FOSOC (see Section B.7). Up to twenty of these files may be processed in one session, and must have the names S1.FRE, S2.FRE, S3.FRE,..... and so on. Furthermore, the Monitor ASSIGN command must be used before running FRAN, since the file(s) will be sought on the device whose logical name is 'RS'. See Section 2.7.2.4 of the RT-11 System Reference Manual. For example, if the 'FRE' files exist on RK05 disk no. 2, the assignment would be:

```
.ASS RK2:RS
.
```

The first question of the dialogue requests the number of files to be processed (1 to 20). The next requests the n-tuple size (4 to 8). The third question requests the number of RAMs per discriminator, whilst answering the fourth with 'Y' causes the program to list (possibly at great length) the frequencies of all of the n-tuple states seen. Any other response suppresses this listing, and gives just a summarising table. The final question requests the total number of patterns processed by FOSOC when creating the FRE file. All program output is sent to FORTRAN logical unit no. 6, which is assigned by default to LP: (the line printer).

There follows a brief description of the output format. For each n-tuple, the program prints the occurrence frequency and identity of every state seen within each class separately, together with the total number of states seen, the number of different states seen, and a detailed list of those states which were seen in patterns of more than one class. The value called 'rejections' is the total number of patterns falling into the multiple states, i.e., those states belonging to more than one class of pattern. This analysis is repeated for every n-tuple, followed by a table summarising the results for each n-tuple.

B.9 ANGELA

This FORTRAN program does not operate under the JANSYS run-time package, but receives all of its commands through an initial dialogue with the user. It is used to process the response files (extension 'RES') produced by MULTI and MULTI2 (see Section B.3.13.6). It may also be used to process the 'RES' files produced by the obsolete programs RECIPE and RECIP4. Up to twenty of these files may be processed in one session, and must have the names SET1.RES, SET2.RES, SET3.RES,..... and so on. Furthermore, the Monitor ASSIGN command must be used before running FRAN, since the file(s) will be sought on the device whose logical name is 'RS'. See Section 2.7.2.4 of the RT-11 System Reference Manual. For example, if the 'RES' files exist on RK05 disk no. 2, the assignment would be:

```
.ASS RK2:RS
.
```

The first question of the dialogue requests the number of files to be processed (1 to 20). The second question requires an option code to be specified: the allowed codes are 1, 2, and 3, whilst any other code causes a reminder to be displayed. Code 1 is the minimum analysis, in which only the header (title, date, classifier parameters, and threshold) is printed, together with the performance figures for the test set (percentages correct, incorrect, and rejected). Code 2 embodies the features of code 1, plus a confusion matrix of the unique classifications (i.e., all decisions made by the classifier, whether right or wrong) and the rejections caused by threshold effects. Code 3 embodies the features of code 2, plus a second confusion matrix showing the multiple classifications, i.e., those patterns for which two or more discriminators gave the maximum response. Note that under a relative threshold greater than zero the latter condition is regarded as a threshold rejection, and thus appears in the first confusion matrix. Threshold rejections are indicated in the first matrix by an entry in the row belonging to the class zero discriminator (having the name '?'). Note that rejection of a class zero pattern is regarded as a correct classification.

The third question in the dialogue requires a 'Y' response if class zero patterns in the response file are to be ignored. Any other response causes them to be processed in the usual way, but there is obviously no

effect if there were no class zero patterns in the test set. The next question requires a 'Y' response if a relative threshold is desired (i.e., in which the maximum response is compared with the next highest). Any other response gives an absolute threshold. The nature of the threshold used is indicated in the results header.

The next question requires a 'Y' response if the threshold originally set in the classifier program (in $10) is to be overridden. If so, three further questions are asked in order to establish the range of threshold values for which the responses are to be analysed, as defined by a lower limit, a higher limit, and an increment size. Any response other than 'Y' to the original question causes the original $10 threshold to be used. The appearance of the message 'STORE FULL BEFORE EOF SEEN. REMAINDER OF FILE NOT USED' during analysis indicates that the response file is too large to be processed by ANGELA, and that ANNY (see below) must be used.

There exists a slightly different version of ANGELA, called ANNY, which must be used if very large response files are to be processed (more than about 5000 patterns). The operational procedures are exactly as for ANGELA, except that class zero patterns are always ignored, and threshold rejections are placed in the second confusion matrix. ANNY functions considerably slower than ANGELA, particularly when several threshold values are being tested for a single response file.

B.10 File structures.

All files are regarded as a series of 8-bit bytes. The value 377 (octal) is frequently used as a marker to indicate the start of a record.

B.10.1 C16

These files are used for storing patterns of up to 16x16 bits, together with a class number, in a fixed record length format with a variable start position. A 377 byte is used to indicate the start of each record - all bytes preceding this at the beginning of the file and between records are ignored. The first 32 bytes of each record (i.e., immediately following the 377) constitute one pattern. The next byte, the last of the record, contains the class number of the pattern, in the range 0 to 255 (although normally only 33 classes are used). The display of such a 16x16 pattern is reconstructed as follows. Bit 0 (least significant bit) of byte 0 (the byte immediately following the 377 marker) is placed at the bottom left of the matrix. Bit 1 of byte 0 goes immediately above this, and so on. Bit 0 of byte 1 thus appears halfway up the leftmost column of the matrix, and bit 0 of byte 2 starts the bottom of the next column along.

B.10.2 RES

These files carry the responses given by MULTI and MULTI2 to test patterns (see B.3.13.6). The format is as follows. The first 4 bytes contain the ASCII characters RES2 for identification purposes. The next byte contains the n-tuple size of the classifier which produced the file, the sixth and seventh bytes contain the X and Y dimensions respectively of the matrix used. The eighth byte contains the year number-72, the ninth contains the day of the month, the tenth the number of the month, the eleventh and twelfth bytes constitute the threshld value ($10), and the thirteenth and fourteenth bytes contain the number of classes used. The next 20 bytes are not used. The next 68 bytes constitute the title given to the

results file, in normal ASCII code. The next 66 bytes contain the names of classes 0 to 32 (two characters per name). The next two bytes are unused. The next byte is the start of the first results record, and thus contains the value 377. Each record contains the discriminator's responses to one pattern, and has the following format. Byte 0 (immediately following the start-of-record marker) contains the response of the class 1 discriminator, byte 1 contains that of the class 2 discriminator, and so on. Byte 32 contains the class number of the pattern producing these results, and is the last byte in the record. It should be noted that the header (n-tuple size, date, title, etc.) occupies exactly 136 bytes, which is the same length as 4 results records of 34 bytes each. RES files are used as input for the analysis programs ANGELA and ANNY.

B.10.3 C64

These files are used for storing 64x64 bit patterns, and have the following format. The first 512 bytes constitute the first pattern (4096 bits), the second 512 bytes the second pattern, and so on. The 64x64 bit display is reconstructed as follows. Bit 0 (least significant bit) of byte 0 in each 512 byte record is placed at the top left of the matrix. Bit 1 of byte 0 goes immediately to the right of this. Bit 0 of byte 1 thus goes to the right of bit 7 of byte 0, and so on. The next row down starts with bit 0 of byte 8, etc.

B.10.4 MMP

These files store the maps required by MULTI and MULTI2 for specifying the way in which the current pattern is transformed into the discriminator input vector. They are not suitable for use with RECIPE or RECIP4. There is a variable record size, each record being a complete map. The number of bytes within a given record is twice the number of RAM inputs of the classifier for which the map was generated. Once the program reading an MMP file has seen the map header, it must read in the correct number of bytes before being able to read the next map. At the start of the file, four bytes are reserved as the header. The first two contain the number of RAM inputs of the

target classifier, and the second two contain the number of pattern elements intended. Theses two values must be equal for a 1:1 classifier.

The significance of each byte within the record is as follows. Firstly, the bytes are combined in pairs to give 16-bit words, i.e., bytes 0 (low order) and 1 (high order) become word 0; bytes 2 (low) and 3 (high) become word 1, and so on. Assume that the RAM inputs (n-tuple elements) of the target classifier are numbered 0 to x (i.e., there are x-1 RAM inputs, and hence (x-1)/n RAMs). Then word 0 contains the pattern element number to which RAM input no. 0 is connected. Similarly word 1 contains the pattern element number to which RAM input no. 1 is connected, and so on. The pattern elements are numbered from the bottom left corner, starting at element 0, going up the first column, then up the second column, and so on. Thus if, for example, word 37 of the map contains the value 153, then pattern element 153 is connected to RAM input no. 37.

B.11 Program examples.

The following examples demonstrate some of the facilities available in each of the programs EDIPIC, MULTI, and HAMAND. For further details, see the appropriate sections above. Note that the ';' symbol appearing in a command string implies that the rest of the command line is treated as a comment.

B.11.1 EDIPIC example.

This sets the matrix dimensions, copies 100 patterns from the file PATIN.C16 to the file PATOUT.C16, forcing the last 20 to become class 6, then obtains 20 patterns of each of classes 1 to 4 from the camera, using the averaging parameters 16 and 10 (see 2.14), and sending them to the new output file TEST35.C16.

```
*X16y16;                                ;Set matrix to 16x16.
*!;                                     ;Open i/p & o/p C16 files.
#,PATOUT<,,patin
*80[G0]20[Gz6o];                        ;Copy 100 patterns: last
*;                                      ;20 class 6.
*!;                                     ;Open new o/p file.
#,TEST35<
*$8=16$9=10;                            ;Set up averaging
*;                                      ;parameters.
*$0=1;                                  ;Count the classes in $0.
*4["Get class "$0:Z$0+q20[N16T10]];     ;Get the patterns.
Get class 1
Get class 2
Get class 3
Get class 4
*
```

B.11.2 MULTI example.

The following example is a program, called CURVY.CMU, which allows several parameters to be set up in a conversational manner, then generates response files, called SET1.RES, SET2.RES, etc., which can be used later to produce a learning curve for the classification system. The program listing is followed by the console output as it would appear, starting at the command (R MULTI) to the RT-11 Monitor to run the MULTI program.

```
; Training curve generation program ----- CURVY.CMU
;
; This is a conversational program which allows
; training curves to be generated from a particular
; training file (edit SPLIT7.C16), and a particular
; testing file (edit TEST2.C16). Parameters to be
; set up during the initial dialogue are: random
; map to be used (obtained from a particular file -
; edit RANMAP.MMP); no. of patterns per class in
; the training and testing files; size of first
; training set; increase in size of training set
; between successive runs; no. of runs (1 to 10 --
; results placed in files SET1.RES .....
; SET10.RES).
;
;
"""This is CURVY.CMU"""
"Training file is SPLIT7.C16"
"Testing file is TEST2.C16"
"Map file is RANMAP.MMP"
"""Random map no. (0 to 3) '$1=?
!
,,RANMAP.MMP
Z1
m$1mq
;
"Actual no. of patterns/class in training file '$6=?
$7=$6,10[$7-]
"Actual no. of patterns/class in testing file '$8=?
"Size of first training set (1 to '$7:") '$0=?
"Increment size (1 to '$0:") '$2=?
$3=$0,$2[$3-]
$4=$6,$0[$4-]
;
"No. of runs (1 to 10) '$5=?
;
!26[$3b$4i$2i]
,SPLIT7.c16
```

```
;
"O.K."
;
; Note that the # causes an ?ILL CMD? message on
; termination.
;
$5[$0:" /class."!26[$3I$2b$4I]!26[$8v]$2[$3+$4-$0+]]#
,SPLIT7.c16
SY2:set1<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set2<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set3<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set4<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set5<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set6<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set7<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set8<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set9<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
,SPLIT7.c16
SY2:set10<,TEST2.c16
$ FROM SPLIT7.C16     TEST: TEST2.C16     RANDOM MAP $
```

The console dialogue is as follows:

```
.R MULTI

MULTI Version 02

Pattern Classifier
```

```
Initial dialogue ?y
Restore old RAM file ?
X Dimension ?16
Y Dimension ?16
N-tuple size ?8
No. of RAMs per discriminator ?
1:1 mapping assumed
No. of discriminators ?26
Rename discriminators ?y
Alphabetically ?y
Absolute threshold ?y
*!
#curvy

This is CURVY.CMU

Training file is SPLIT7.C16
Testing file is TEST2.C16
Map file is RANMAP.MMP

Random map no. (0 to 3) =2
Actual no. of patterns/class in training file =70
Actual no. of patterns/class in testing file =5
Size of first training set (1 to 60) =5
Increment size (1 to 5) =4
No. of runs (1 to 10) =5
O.K.
5 /class.
9 /class.
13 /class.
17 /class.
21 /class.
?ILL CMD?
.ki

.
```

B.11.3 HAMAND example.

In this example, the average patterns are built up using the file AVGPAT.C16, then the file PATOUT.C16 is created from the file PATIN.C16 by selecting only those patterns which are within the Hamming distance range of 50 to 100 from the average patterns of the appropriate classes. See B.5.14.

```
*QC;                                    ;Reset everything.
*!;                                     ;Open input file.
#,AVGPAT
*2600[GA];                              ;Build up average
*;                                      ;patterns.
*!;                                     ;Open o/p & i/p files.
#,PATOUT<,PATIN
*$8=50,$9=100;                          ;Set Hamming distance
*;                                      ;limits.
*;                                      ;Output all patterns
*;                                      ;within the
*;                                      ;limits: note that the
*;                                      ;terminating
*;                                      ;error condition is
*;                                      ;unavoidable.
*10000[WO];                             ;No data is lost.
?I/O EOF ERR?   CHANNEL 4
.
```

B.12 Error message summary.

The following is a list of the error messages possible under JANSYS and any of the programs currently using JANSYS.

I/O EOF ERR	End of file reached on a read or write operation.
I/O DEV ERR	Hardware error detected on read or write, e.g, a write protected device was specified for output.
CH CLOSED	Attempt to use a channel which has not been assigned to a device/file combination.
NO HANDLER	Attempt to use a device whose handler was not LOADed in core prior to running the program.
ILL DEV	Attempt to use an unsuitable or non-existent device.
FIL NOT FND	Requested file does not exist on specified device.
DEV FUL	No room on output device (may also occur if device is write-protected).
ILL CMD	Bad file specification syntax, or unrecognized command symbol.
ILL TTY	Attempt to assign TT: to an unsuitable channel.
BAD CH NO	Programmed attempt to send unsuitable data to TT:.
OUT ERR	Error detected when trying to close an output file ready for a new assignment. Loss of data may occur.
VT11 ERR	Error detected in display file. Repeated CTRL/C may be necessary for recovery. If this fails, reboot the system.
BAD DATA	Numerical datum was out of range.
BREAK IN	Response to CTRL/C from console keyboard.

RESTART	Response to RE command after CTRL/C.
LOOP COUNT	Square brackets in command string do not match, or nesting level exceeds 5.
NO MAP	Attempt to read a pattern before loading a map. Note that MULTI and MULTI2 do not have default mappings.
BOX OFF	Attempt to use box edit commands (/ or \) in EDIFIC whilst the program is in 'window' mode. See Section B.2.15.
INSANE USER	Frivolous comment arising when the user is not sure whether he wants to re-configure the system in FOSOC.
NO CURRENT N-TUPLE	Attempt to use single n-tuple commands (D, E, or S) when there is no current n-tuple in MAPFIT (see Section B.6.7).
NO FRAMES	Attempt to dump an accumulated pattern in HAMAND when no patterns have been seen.
CLASS ZERO	Attempt to dump a class zero accumulated pattern in HAMAND.
BAD MAP	Map file has unsuitable format (may be because matrix dimensions are incorrect).
CLASS NO.	Attempt to train discriminator no. 0. Attempt to access average pattern for class 0 --- HAMAND only.
M-SWAP ERR	Issued by the RT11 Monitor, usually when write protection of the system disk has prevented a system swapping operation. Cure: write enable the system disk and try again.

A number of self-explanatory error messages can occur during the dialogue phases of the various programs.

B.13.1 EDIFIC command summary - Version 04.

Command	Section	Action
!	B.2.1	Performs channel assignments.
X	B.2.2	Set X dimension.
Y	B.2.2	Set Y dimension.
G	B.2.3	Get new 16x16 pattern from channel 5.
I	B.2.4	Skip a 16x16 pattern on channel 5.
Z	B.2.5	Set new class number.
H	B.2.6	Move window left.
J	B.2.6	Move window down.
K	B.2.6	Move window up.
L	B.2.6	Move window right.
F	B.2.7	Get new 64x64 pattern from channel 4.
C	B.2.8	Get new 64x64 pattern from camera.
W	B.2.9	Output 64x64 pattern on channel 0.
O	B.2.10	Output 16x16 pattern on channel 1.
M	B.2.11	Output map on channel 2.
P	B.2.12	Output 16x16 ASCII on channel 2.
D	B.2.12	Output 64x64 ASCII on channel 2.
N	B.2.13	Set frame count for averager.
T	B.2.13	Set threshold for averager.
Q	B.2.14	Get C64 frames until CTRL/Y interrupt.
A	B.2.15	Set/reset box mode.
/	B.2.15	Set enclosed bit.
\	B.2.15	Clear enclosed bit.
R	B.2.15	Clear window & reset box position.
S	B.2.15	Reset box position.

B.13.2 MULTI/MULTI2 command summary - Versions 01/01.

Command	Section	Action
!	B.3.2	Perform channel assignments.
G	B.3.3	Get new 16x16 pattern from channel 4.
I	B.3.4	Skip a 16x16 pattern on channel 4.
Z	B.3.5	Set new class number.
H	B.3.6	Move window left.
J	B.3.6	Move window down.
K	B.3.6	Move window up.
L	B.3.6	Move window right.
C	B.3.7	Get new 64x64 pattern from camera.
T	B.3.8	Teach current class on current pattern.
B	B.3.9	Combined G and T commands.
Q	B.3.10	Clear all discriminators.
M	B.3.11	Read one map from channel 5.
D	B.3.12	Dump one RAM from current discriminator.
X	B.3.13.1	Update maximum response.
Y	B.3.13.2	Print class number and maximum response.
P	B.3.13.3	Print discrim. responses to current pat.
A	B.3.13.4	Print RAM responses to current pattern.
E	B.3.13.5	Collate recognition performance ($8, $9).
S	B.3.13.6	Output responses on channel 0.
V	B.3.13.7	Combined G and S commands.
W	B.3.14	Train if response incorrect.
O	B.3.15	Save discriminators on channel 1.
R	B.3.16	Stop on first incorrect response.
F	B.3.17	Perform window 'tracking'.
U	B.3.18	Unteach on current pattern.
N	B.3.19	Unteach if response incorrect.

B.13.3 HAMAND command summary - Version 02.

Command	Section	Action
!	B.5.1	Perform channel assignments.
G	B.5.2	Get new 16x16 pattern from channel 4.
I	B.5.3	Skip a 16x16 pattern on channel 4.
Z	B.5.4	Set new class number.
O	B.5.5	O/p current 16x16 pattern on channel 1.
R	B.5.6	Reset accumulators for current class.
C	B.5.7	Reset all accumulators.
A	B.5.8	Accumulate current pattern.
V	B.5.9	Display avge. pattern for current class.
X	B.5.10	Display acc. pattern for current class.
Y	B.5.11	O/p avge. pattern in ASCII on channel 2.
E	B.5.12	O/p class membership of file on chan. 4.
H	B.5.13	Display H. distance of current pattern.
W	B.5.14	Search file on channel 4 for pattern.
Q	B.5.15	Clear histogram accumulators.
S	B.5.15	Add H dist. of current pat. to histogram.
T	B.5.15	Output histogram on channel 2.
P	B.5.16	O/p average binary pattern on channel 1.
J	B.5.17	O/p H. distance in text on channel 2.
B	B.5.18	O/p accum. pattern in text on channel 2.
D	B.5.19	O/p accum. binary pattern on channel 0.

B.13.4 MAPPIT command summary - Version 02.

Command	Section	Action
!	B.6.1	Perform channel assignments.
C	B.6.2	Re-configure system.
M	B.6.3	Generate random map.
R	B.6.4	Generate regular map.
O	B.6.5	Output complete map on channel 0.
I	B.6.6	Input complete map from channel 4.
N	B.6.7	Set current n-tuple number.
X	B.6.8	Restore full map mode.
E	B.6.9	Redefine one bit of current n-tuple.
D	B.6.10	Redefine whole of current n-tuple.
P	B.6.11	O/p map display in text on channel 1.
H	B.6.12	O/p map header on channel 0.
S	B.6.13	O/p current n-tuple on channel 0.

B.13.5 FOSOC command summary -- Version 01.

Command	Section	Action
!	B.7.1	Perform channel assignments.
C	B.7.2	Re-configure system.
Q	B.7.3	Clear all accumulators.
M	B.7.4	Read in MMP map from channel 5.
G	B.7.5	Read next pattern from chan. 4.
I	B.7.6	Skip next pattern on channel 4.
A	B.7.7	Accumulate current n-tuple states.
S	B.7.8	List states seen on channel 1.
F	B.7.9	O/p accums. in binary on chan. 0.

References

1. Plummer,J & Meindl,J. Portable Reading Aid for the Blind. IEEE Trans. SC-7, 1972. p.111.

2. Nye,P & Bliss,J. Sensory Aids for the Blind. Proc. IEEE 58, 1970. p.1878.

3. Linvill,J & Bliss,J. A Direct Translation Reading Aid for the Blind. Proc. IEEE 54. 1966. p.40.

4. Beddoes,M & Suen,C. Presentation of Sound Output from the Lexiphone. IEEE Trans. BME-18. 1971. p.85.

5. Freiberger,H & Murphy,E. Reading Machines for the Blind. IEEE Trans. HFE-2. 1961. p.8.

6. Lee,F. Reading Machines: from Text to Speech. IEEE Trans. AU-17. 1969. p.275.

7. Smith,G. Cognodictor. Proc.23rd Annual Conference on Engineering in Medicine and Biology. 1970. p.167.

8. Mason,S & Clemens,J. Character Recognition in an Experimental Reading Machine for the Blind; in Recognizing Patterns, by Kolers and Eden. MIT. 1968.

9. Nye,P. Reading Aids for the Blind. IEEE Trans. BME-17. 1970. p.97.

10. Cooper,F. Reading Aids for the Blind. IEEE Trans. Au-17. 1969. p. 266.

11. Uttley,A. Conversion of Printed Characters into Speechlike Sound. Medical and Biomedical Engineering. March 1975.

12. Greenshields, I.R. Some Aspects of the Interactive Evaluation of a Blind Reading Aid Output. PhD thesis. Department of Electrical Engineering and Electronics, Brunel University. 1977.

13. Ullmann, J.R. Pattern Recognition Techniques. Butterworths, 1973.

14. Bledsoe, W.W. and Browning, I. Pattern Recognition and Reading by Machine. Proc. EJCC, Boston. 1959.

15. Highleyman, W.H. and Kamentsky, L.A. Comments on a Character Recognition Method of Bledsoe and Browning. IRE Trans.EC-9 Correspondence. 1960.

16. Bledsoe, W.W. and Bisson, C.L. Improved Memory Matrices for the n-Tuple Pattern Recognition Method. IRE Trans. EC-11. June 1962.

17. Highleman, W.H. Further Comments on the n-tuple Pattern Recognition Method. IRE Trans. EC-10. Correspondence. 1961.

18. Minsky, M.L. Steps Toward Artificial Intelligence. Proc. IRE. 49. p.14. 1961.

19. Steck, G.P. Stochastic Model for the Browning-Bledsoe Pattern Recognition Scheme. IRE Trans. EC-11. 1962.

20. Roy, R.J. and Sherman, J. Two Viewpoints of k-tuple Pattern Recognition. IEEE Trans. SSC-3. 1967.

21. Ullmann, J.R. and Kidd, P.A. Recognition Experiments with Typed Numerals from Envelopes in the Mail. Pattern Recognition 1. July 1969.

22. Ullmann, J.R. Experiments with the n-tuple Method of Pattern Recognition. IEEE Trans. C-18. 1969.

23. Ullmann, J.R. Reduction of Storage Requirements of Bledsoe and Browning's n-tuple Method of Pattern Recognition. Pattern Recognition 3. 1971.

24. Aleksander, I. and Albrow, R.C. Pattern Recognition with Adaptive Logic Circuits. IEE/NPL Conference on Pattern Recognition. 1968.

25. Aleksander, I. Microcircuit Learning Computers. Mills and Boon. 1971.

26. Stonham, T.J. Improved Hamming-Distance Analysis for Digital Learning Networks. Electronics Letters. 13, No.6. 1977.

27. Reeves, A.P. A Digital Learning System for Tracking Pattern Features. PhD thesis. University of Kent at Canterbury, 1973.

28. Cheung-Yueng-San, C.Y.E. Some Aspects of Adaptive Logic for Pattern Recognition. PhD thesis. University of Kent at Canterbury. 1973.

29. Stonham, T.J. The Classification of Mass Spectra with Adaptive Logic Networks. PhD thesis. University of Kent at Canterbury. 1974.

30. Aleksander, I., Stonham, T.J. and Wilson, M.J.D. Adaptive Logic for Artificially Intelligent Systems. Radio and Electronic Engineer. 44.1974

31. Aleksander, I. and Hanna, F.K. Automata Theory: An Engineering Approach. Edward Arnold, London. 1976. Cahpter 5.

32. Fairhurst,M.C. and Stonham,T.J. A Classification System for Alpha-Numeric Characters. Digital Processes. 2. 1976.

33. Williams, P. Some Experiments Using n-tuple Techniques for Alphanumeric Pattern Classification. Post Office Research Department Report 575. 1977.

34. Prorok, J.G. Networks of Logic Elements in Decision Making with Many Binary Inputs. PhD thesis. Queen Mary College, University of London. 1976.

35. Stonham,T.J. and Aleksander,I. Optimisation of Digital Learning Networks when Applied to Pattern Recognition of Mass Spectra. Electronics Letters 10, No.15. 1974.

36. Mistry, K.D. Reduction of Interference in Video Signals from a Solid State Camera System. Undergraduate project report, Department of Electrical Engineering and Electronics, Brunel University. 1976.

37. Fry, Noble, and Ryecroft. Fixed Pattern Noise in Photomatrices. IEEE Trans. SC-5. 1970.

38. Digital Equipment Corporation. RT-11 System Reference Manual. Order No. DEC-11-ORUGA-C-D. 1976.

Acknowledgements

I would like to thank my superviser, Professor Igor Aleksander, for his invaluable advice and encouragement throughout the project.

I am also grateful to the Science Research Council who provided financial support.

Finally my thanks are due to Mrs.Joan Hurley, who typed the thesis.

John Nappey